

Final Project

Autonomous Underwater Vehicle

Richard Sefton

September 5, 2024

Contents

Introduction	1
Literature Review	3
Previous Works	3
Amateur Level	3
Commercial Level	3
Research Level	4
Key Literature	4
Project Design	8
Domain	8
Technologies	9
Microcontrollers	9
AUV Body	9
Navigation	10
Mechanical	11
Implementation	12
I^2C	12
Sonar	14
Depth Controller	15
Appendices	22
1.1 AVR ATTiny 1627 Features	22
1.2 VQFN Breakout	23
1.3 AUV Electronics Schematics	26
1.3.1 AUV Schematic	26
1.3.2 Central Controller Schematic	27
1.3.3 Depth Controller Schematic	28
1.3.4 Sonar Module Schematic	28
1.3.5 Main Motor Schematic	29
1.3.6 Side Motor Schematic	30
1.4 Pseudocode	30
1.4.1 Central Controller	30

1.4.2	Depth Controller	32
1.4.3	Main Motor	34
1.4.4	Sonar	35
1.4.5	Side Motor	36
1.5	Midterm I^2C	36
1.5.1	host.h	36
1.5.2	host.c	37
1.5.3	client.h	40
1.5.4	client.c	40
1.6	CWire Library	45
1.6.1	CWire.h	45
1.6.2	CWire.c	52
1.7	AUV Designs	66
1.8	Final Code	97
1.8.1	Common ShortTypes.h	97
1.8.2	Common Common.h	97
1.8.3	Common Common.c	97
1.8.4	Central Controller AddressBook.h	98
1.8.5	Central Controller AUV.h	98
1.8.6	Central Controller AUV.c	100
1.8.7	Central Controller CentralController.h	102
1.8.8	Stepper Motor StepperMotor.h	109
1.8.9	Stepper Motor StepperMotor.c	110
1.8.10	Depth Controller DepthController.h	112
1.8.11	Sonar Sonar.h	116
1.8.12	Sonar Sonar.c	117
1.8.13	Sonar SonarModule.h	118
1.8.14	Main Motor MainMotor.h	120
1.8.15	Side Motor SideMotor.h	122

Abstract

This report depicts the end to end design and development of an Autonomous Underwater Vehicle for the Final Project of the BSc Computer Science degree. As it transpires this was an incredibly ambitious project to deliver in such a short time frame. This report will try to capture the key aspects of the development process which will be a challenge in itself given the constraints on the report and as such, some aspects will be focused on more than others.

Introduction

(max 2 pages) Autonomous Underwater Vehicles have been around since the 1950s, with the first AUV on record developed by Washington University in 1957 named SPURV[10] (Self Propelled Underwater Research Vehicle). The device was fitted with various temperature and pressure sending probes and could traverse to depths of 3600m and one of the last uses of the SPURV AUV was to study submarine wakes in the 70s. Since this first development, in the last 67 years AUVs have been refined and further developed by various governments, oceanographic institutes, universities and private companies. Today, AUVs are often considered cheaper, better and safer to deploy than manned underwater vehicles (particularly in the light of the more recent Titan submarine implosion[22]), and safer than deploying divers. They are capable of obtaining more data than any single diver can obtain, and capable of diving longer than any manned vehicle as the requirement for breathable air is negated. An AUVs dive time is only limited by its available power which will only increase as power technology improves. AUVs are used across a multitude of different industrial sectors spanning scientific research, surveying, military, shipping and gas and oil[11].

The aim of this project is to design and develop a rudimentary AUV. This project does not comfortably fall into any of the predefined template project ideas but was born from a personal enjoyment of the Internet of Things module which allowed me to explore microcontroller programming and interacting with physical sensors and actuators, which blossomed into a passion for robotics. As one of the early module videos highlighted, our projects should be something we are passionate about as without passion we would lack the motivation to complete such a lengthy and isolated piece of work[24]. I found that the existing Internet of Things templates involved topics I'm not particularly passionate about. I did gain permission from the Coursera Tutor Forums prior to embarking on this journey[30].

This project was initially inspired by the work of a YouTuber (Brick Experiment Channel[3]) who created a small remote controlled submarine using a watertight container and Lego. The accompanying blog[4] depicts the design and development of several versions of this device, and also explains some of the key concepts that need to be considered such as buoyancy. The original plan for this project was to do something similar to this (occupying

the hobby space of the AUV market) - a small device that provides basic functionality autonomously, using similar materials (a water tight container and Lego). However, following the initial research into the AUV market there is a trend of AUVs costing thousands[23], a cost which will increase with the additional custom payloads (sensor arrays) required for the specific research projects. There are a small handful of manufacturers that offer lower cost AUVs to customers such as the ecoSub[5] which states one of their submarines are \approx £15k, this is a price point that may still be out of reach for many projects. In addition to the initial cost of AUVs being expensive, the parts would be custom made by the manufacturer and so any repairs could be equally costly.

While the inspiration for this project started with a small remote controlled submarine, the initial research has expanded the scope to try and develop an extremely low cost AUV which could be made open source (allowing for easy customisations) and can be manufactured by the researchers that require them using readily available parts and 3D printing which is now common relatively common and would mitigate the requirement for 3rd party companies fabricating parts. This would also mitigate costly repairs as all the design files would be a part of the project. The target users of this AUV may be required to develop their own payloads, but even the cost of hiring a developer to do this may still be cheaper than buying AUVs from known sources.

By developing an open source AUV, this could seriously open up underwater research for projects that are underfunded. These projects may have previously relied on human divers to obtain data as the safer unmanned alternative is out of reach for them so this would make such endeavours safer too.

Literature Review

(max 5 pages)

Previous works

Amateur Level

The YouTuber Brick Experiment Channel[3] wrote a series of blog posts[4] outlining the build process for the 4th iteration of their remote controlled submarine.

The posts are detailed covering the steps of building this submarine with reasoning behind the design decisions along with a critical evaluation of each step. This version is using a large syringe to bring water onboard the vehicle to control buoyancy, and is driven by two motors: one for propulsion, and one for direction. The sensors incorporated measure pressure to accurately gauge depth, and a laser to measure distance from the bed of a body of water.

The blog also raised communication as a consideration: While the aim of this project is to build an autonomous vehicle, it would be nice to communicate with it in flight. The problem is this cannot be done easily with wireless frequencies as the higher frequencies that provide the required bandwidth do not penetrate water that well.

Commercial Level

The company Advanced Navigation makes the Hydrus Drone[21], which claims to be one of the smallest AUVs on the market. The device itself controls depth and position with impellers, a 4K camera with AI integrations, and a forward facing sonar. The most important aspect of Advanced Navigation's AUV is how they address the issues around wireless communication. This company has fitted their device with acoustic and optical modems so data can be transferred using audio frequencies and light (presumably similar to fibre optics).

Research Level

The Woods Hole Oceanographic Institute[13] draws attention to the use of different AUVs depending on the area of research or environment. They use one of 6 AUV models, which operate at different depths or different functionality. The REMUS[14] can be equipped with varying sensors and is programmed for survey missions. It was also adapted to survey the Delaware river Aqueduct for leaks.

One of the more interesting AUVs this institute uses is the Spray Glider[15]. This description for this device is one that glides through the water without any external thrusters so can be travel for weeks at a time. The device uses internal bladders to control buoyancy, and is set to navigate preset paths equipped with varying sensors. What is interesting about this device is the lack of external propulsion seems to make this device more passive, and can therefore operate at lower voltage.

The UK National Oceanography Centre has details on the challenges around an designing an AUV[12]. For power and propulsion it highlights the lack of oxygen for internal combustion makes it necessary to use batteries for power and notes the difference in speed between AUVs and surface ships. For navigation it highlights that because GPS can't penetrate the top few mm of water, the reliance on other techniques is required. It suggests an approach called Dead Reckoning which gets the speed of the AUV by measuring the Doppler shift from the sea bed for relative speed and using a gyro to measure the heading. It highlights that navigational accuracy is vital to survey missions.

Key Literature

The definition for an AUV is best described in an article titled "Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience".

Autonomous Underwater Vehicles (AUVs) are unmanned, self-propelled vehicles that are typically deployed from a surface vessel, and can operate independently of that vessel for periods of a few h to several days.[36]

The authors go on to highlight that "In addition, recent economic drivers, such as rapidly increasing vessel fuel oil costs, are making autonomous systems a potentially attractive proposition to organisations responsible for large-scale and cost-effective marine data collection programmes"[36]. This article addresses the commercial benefits of AUV production and deployment but doesn't address the human benefits such as safety.

The authors also discuss the applications of AUVs and point out that "the sensors deployed determine the vehicle altitude, as well as its speed

and endurance.”[36] The endurance in this context is the available power to the AUV which would affect its overall range. Because different sensors and actuators have varying power requirements, some of these components may draw more power than others. The array of sensors required is defined by the AUVs use case: it would be inefficient mounting water quality sensors onto an AUV designed for ocean bed mapping survey.

While this article includes a lot of jargon relating to marine geosciences, it provides a good overview of the uses of AUVs and their previous applications. More importantly, it provides some good starting points for some concepts I will need to explore during the development of this project. This includes references to works that look into the Dead Reckoning method of positioning, and a launch pad into some key concepts such as Sonar.

Because the sensors deployed on an AUV are specific to its function and is something that can be configured, it is worth considering a modular design. As a part of this there could be some level of connectivity between the separate modules. Modern MCUs typically feature multiple options for communication protocols such as UART, USART, I^2C , SPI, USB, Ethernet or could be as simple as pulling a pin high or low. For the purpose of this project, we will consider I^2C as it uses minimal wires, and allows for two way communication. The MCU data sheet[20] is the manufacturers documentation and is a vital reference for development on any given MCU and it directly relates to the MCU specific library in code. It has all the details and requirements to setup any of the provided MCU features, and gives a brief overview of each feature.

With regards to the ATTiny 1627 MCU, the data sheet describes I^2C as a TWI (Two Wire Interface): “The Two-Wire Interface (TWI) is a bidirectional, two-wire communication interface (bus) with a Serial Data Line (SDA) and a Serial Clock Line (SCL)”[20]. There are several advantages to using the TWI bus as a communication protocol as it allows the connection of “one or several slave devices to one or several master devices”[20] allowing a network of devices to become interconnected. This would also allow for parallel computation as instead leaving one device to perform all of the sensor readings and calculations, some of which could be computationally expensive or using deliberate pauses of code execution, a master device could simply trigger the networked modules to each perform their actions and then do nothing while waiting for the responses.

In the article ‘FEATURES, OPERATION PRINCIPLE AND LIMITS OF SPI AND I^2C COMMUNICATION PROTOCOLS FOR SMART OBJECTS’ the authors share their opinion on how an IoT network should operate and how it should be secured, and then give an overview of both I^2C and SPI protocols before suggesting a hybrid version of the two. For IoT networks the authors suggest that “different devices with different capabilities will take part in the creation of a network. A self-describing interface for each device is necessary in order to optimize the management of required

tasks”[33]. This point backs up my idea of a modular design where sensor modules can be attached in a way to form a network where each module is responsible for a specific task. They then go on to highlight that “adding an object to the network should not cause the collapse of the network itself. The network should also properly handle the failure of a device”[33]. This point involves error handling within a network of discreet modules. If one module fails, the whole unit should not. If the module is deemed vital to the devices operation, such as obstacle avoidance sensors on an AUV, a graceful way to handle this could be to surface the device and broadcast a signal indicating the error on a predefined wavelength so it can be collected.

For I^2C the authors give a detailed overview stating that:

The *Inter Integrated Circuit* (I^2C) protocol, ..., was developed by Philips in 1982 and it is a serial, single-ended bus with multi-master support and typically used to connect low speed devices.[33]

The bandwidth available over the standard I^2C rail is slow at 100kbit/s, but there is a Fast mode and Fast Plus mode which allows up to 1Mbit/s. Additionally some devices also support a high speed mode which allows up to 3.4Mbit/s.[2]. This means the I^2C rail may not be suitable for *all* types of sensors attached, particularly ones that gather a lot of data in real time, for example high definition cameras.

Since the bus is completely shared, it is possible, for devices, to receive not only unicast transmission but also broadcast messages: for this reason, whenever master wants to write or read data from a particular slave, it will address it first. Addressing bits were originally seven, extended to 10 with the latest reviews to increase supported maximum number of connected devices. I^2C bus supports multiple masters on the same bus too, but a proper conflict-solving algorithm has to be implemented. [33]

This addressing is how the devices know what devices they need to talk to on a network. With the system of addressing and the ability to broadcast transmissions we can begin to visualise the I^2C protocol as akin to web sockets. The interesting point raised here is that conflict resolution needs to be handled where the network has multiple masters. One such proposal they put forward is Arbitration: “since I^2C supports multiple masters, an arbitration rule is necessary. In this protocol, arbitration proceeds bit by bit and the rule is deterministic: the first master that produces a one when the other produces a zero loses the arbitration”[33].

Another resource that covers a lot of information about AVR Microcontrollers and also providing an overview of the features as well as how to

interact with actuators such as servos and stepper motors, is 'Make: AVR Programming'[35]. While this book does contain quite a broad overview of the features and protocols, it also has examples of code to interact with them, as well as an overview of the toolchain required to programme a microcontroller (in this instance involving Make, avr-gcc[8] and AVR Dude[1] as well as a physical hardware programmer). However, the example code is all written for a specific MCU that the book focuses on, being the AT-MEGA168. However, as the author notes, this should not be too difficult to port to my specifically chosen MCU as the book also tries to teach another skill being the ability to refer to relevant sections of the data sheet.

As well as internal device communication protocols, it would also be good if we could have some way of interacting with the device in flight: receiving data, and possibly some rudimentary manual over ride/remote control. As corroborated in an article exploring underwater wireless networks, "radio is seriously attenuated in water"[9]. Radio waves exist on the EM Spectrum and the lower the frequency, the further it can penetrate water due to the electrical dissipation that occurs, which is accentuated by salt water. High frequencies that we are used to as standard data transfer frequencies (WiFi, Bluetooth) are measured in GHz, and these penetrate only a few cm of water. Even LoRa at 433MHz will only penetrate 30-50cm of water. The military use lower radio frequencies operating in the range of Hz to kHz to communicate with their submarines but the trade off is lower frequencies bandwidth (less data can be transferred as there are fewer cycles per second). Additionally, radio frequencies are controlled[25] and as a hobby enthusiast/student I only have access to a limited range of frequencies.

The article also states "acoustic communication is almost the only effective way for underwater wireless transmission. Nevertheless, compared with the traditional radio, underwater acoustic communication is greatly affected by poor conditions, such as absorption, scattering, multipath interference and Doppler effect"[9]. While I think the authors are perhaps exaggerating this point to justify their experiments somewhat, it does offer an alternate avenue to wireless communication with a device while it is operating underwater, but it would require the development of an ultrasonic modem of some description. Alternatively we could just add a plug to the device for a physical wire for external control/data transmission. While this goes against the design of an AUV as being untethered, it would also act as a lifeline to retrieve the device while minimising risk, even when testing in controlled bodies of water.

Design

(max 4 pages)

Domain

AUVs have applications spanning multiple domains both in scientific exploration as well as commercial enterprise. For science they are used for surveying coastlines and ocean beds, sampling in remote or hard to reach locations and exploring the depths to name a few applications. For commerce they can be deployed to monitor gas and oil pipelines, drills, survey potential resource sites, and monitor ship hulls etc. The deployment of AUVs over physical diving or manned equipment offer more benefits when compared to the cost and safety. AUVs can potentially spend longer underwater depending on the design and payload they are equipped with and the size of the power source, and can gather far more data than a diver with far less risk to human life.

For this project, the aim was to develop a low cost, open source AUV that will be targeted towards the scientific exploration domain. Since this domain still covers a multitude of deployment scenarios (coastal surveys, lake/ocean floor mapping, deep sea exploration) spanning countless research applications including water quality/temperature, imaging/mapping, ecosystem exploration etc., this project will initially target scenarios that require an AUV to more likely to dive to shallower depths such as coastal surveys. Since the sensor payloads vary depending on the research requirements, the plan is to deliver the AUV shell that can perform the basics of autonomous traversal of a body of water relying on sonar technology for obstacle avoidance which can be easy to extend relying on the I^2C bus for communication of connected modules. This can be extended by the researchers deploying the AUV either by integrating with the main electronics driving the device, or by making their own sensor payload independent to the main electronics.

Technologies

Microcontrollers

Because this project is something to be deployed to remote environments, the microcontrollers required should be efficient and low powered. To achieve this I chose the ATTiny 1627 microcontroller mentioned previously in this report. This micro controller has been chosen due the fact it is feature rich with low power requirements as the device itself can operate from 1.8V to 5.5V and the efficiency can potentially be extended using the devices sleep settings. The full list of features for this microcontroller can be found in the appendices 1.1. Because this chip is extremely small, with the 24 pin version only being available in a QFN/VQFN (Quad Flat No leads) packaging, it was necessary to develop a simple breakout board to make this chip breadboard compatible. This required a small amount of learning how to design PCBs, but using KiCad[16], this proved relatively easy for such a simple design, and after manufacturing costs and MCU costs this proved economical as each board cost \approx £8 which is a price comparable to the popular Node 8266 MCU used during the Internet of Things module. The designs for this board can be seen in the appendices 1.2

While device packs such as the megaTinyCore[18] would allow the development of Arduino code on these microcontrollers, the chosen approach was to use “bare metal” C code and directly use the core AVR libraries to interact with the various registers. This is because, while Arduino as a level of abstraction has various API methods that make development a bit easier, it comes with a loss of being able to easily configure many of the device peripherals to be used in specific scenarios, and some of the peripherals (such as Timer Counters) are already configured and used for some functions such as delay. By relying on peripherals and ISRs(Interrupt Service Routines), the code can be developed to be more event driven than running code in a main loop. ISRs work by raising an interrupt flag on the peripheral once a condition has been met. The flag triggers any code inside an ISR vector to be run. The design for the modules will be several modules interconnected using the I^2C /TWI protocol in a star topology where only one device is the Host and the rest are Clients, and are designed in such as way that the pseudocode for each of the modules found in the appendices 1.4 can be used as a starting point ot develop the bare metal C code to run on the microcontrollers.

AUV Body

In order to keep the AUV as accessible as possible to any institutes or researchers looking to develop one, the AUV hull and internal fixtures has been designed to be 3D printed. This negates the requirement for users to need to get the parts fabricated in materials that require special tools or

specialised knowledge on how to work with them, or that are costly such as metal or fibreglass or carbon fibre. Once the initial investment of a 3D printer is overcome if not already available (\approx £400 for one that prints a variety of materials), the cost in filament is extremely low in comparison. This approach does require further research and experimentation to determine the best combination of filaments or materials to print the components in, along with any potential post-processing to increase the suitability and durability of the parts. Since it is common knowledge that often electronics and water do not mix, the design used magnets to connect moving parts on the inside to the outside of the vessel (propellers), as this removes the need to try and have a hole with enough clearance to freely rotate a shaft, while keeping the vessel waterproof.

Navigation

For navigation and obstacle avoidance this project employs the use of Sonar. Sonar specifically for underwater applications on the consumer level is extremely hard to come by but there are some options available such as the Ping2[29]. while these are feature rich (using multiple frequencies for accurate sonar imaging), they are quite expensive (\approx £400), and future availability could be questionable. To this end, the original plan starting this project was to try to develop a custom sonar module using transducers rated for underwater use[27] that would operate by providing a distance in cm upon request. The requesting trigger would be by command via the I^2C rail. The distance would then be available on the next I^2C read request. This could have also been developed further to having an additional pin to notify the host device that the distance is ready to be requested, but really would be unnecessary given how fast the speed of sound travels, particularly underwater (1500m/s).

This however proved to be a fruitless endeavour which I can only surmise was down to a combination of inexperience with analogue/audio electronics and not meeting the voltage requirements of the transducers (160Vpp). This ended up using significantly more of the planned time than expected and in the end depended on the use of prefabricated ultrasonic rangefinders rated as “waterproof” (JSN-SR04T[32]). These devices also work in a similar way to the HC-SR04[7] where the trigger pin is pulled high for a minimum of 10 μ s then released which triggers the transmitting transducer to pulse, and pulls the echo pin high. Once the receiving transducer receives a signal, the echo pin is pulled low and the time between its high and low state can be used to calculate the distance using the formula:

$$distance = \frac{(time * speedOfSound)}{2}$$

The division by 2 in this formula is because we only need to calculate

the distance to the object, and the pulse measured is the time it takes the ultrasonic waves to reach the object and then return.

Mechanical

The responses of the sonar modules incorporated into the design of this AUV are used to drive actuators to respond to the physical environment. This includes having the ability to dive and rise, move forward, backwards, and rotate to a new heading to avoid obstacles. With the aim of keeping this device low cost, the goal was to attempt to use inexpensive, readily available parts (stepper and toy motors). The Stepper motor was chosen as it provides precise control over some parts (the depth controller) and can also provide enough torque to physically operate them (depth controller is a large syringe connected to a threaded shaft to convert the rotary force into a linear motion). The main propeller was also quite large and so benefited from this additional torque. The toy motors chosen to turn the smaller side motors as these would benefit from the raw speed they can provide.

Implementation

(max 3 pages)

I^2C

The main lynch pin of this project is the I^2C protocol used to connect all the individual modules of this vehicle. There are a few different registers on the MCU that require writing to in order to set up the protocol correctly depending on if you want to configure it as a host, a client, or both and numerous considerations when developing a library to handle this protocol effectively as the bus used for communication can enter into a variety of different states.

First, the device data sheet[20] provides a very good overview of what is required to initialise the protocol and use it for effective communication. The initialisation procedures and further instructions are many pages long as it tries to be as thorough as possible to cover all the different scenarios (covering read, write operations, bus idle and error states and arbitration rules and how to handle these along with various interrupt flags that can be raised). This was the original point of reference for this protocol which was used to develop my own library from scratch (appendices 1.5) with the library code split into code specifically for the host device and client device depending on the specific use case of each MCU. For the host (appendix 1.5.1 and 1.5.2), the initialisation process involved setting the baud rate, setting the bus status to 'IDLE', and because I decided to use the peripherals "Smart mode" to save myself from having to manually send ACK and NACK bits after each byte is transmitted/received, the MCTRLB register has to have the relevant ACKACT configuration set, and then the protocol is enabled by writing to the MCTRLA register. Using the protocol was theoretically simple, with commands initiated by the host. The host issues a command to a connected client device by first sending the address by writing to the MADDR register (7 bits) which is shifted left by 1 bit as the first bit is used to indicate if the operation is read(0) or write(1). Once the client has responded with ACK (acknowledging it has received the request and is ready to receive more data), either the host transmits data until the client sends

a NACK bit (indicating it can't receive or won't accept any more data), or the client sends data till the host sends an ACK or NACK bit.

The client code (appendices 1.5.3 and 1.5.4) initialisation is slightly simpler. For this, we simply set the address of the client to the SADDR register (7 bits shifted left by 1 bit), with the first bit indicating if the protocol should respond to open broadcasts on the bus (messages addressed to 0x00), and also the SCTRLA register is where the protocol is enabled and the Interrupts are enabled. The client uses interrupts and callbacks to handle the operations. The ISR code first checks the if the Address or Stop interrupt flag (APIF) is raised which if this is the case, it will check the address match flag (AP). If AP flag is raised this is the Address match and so it would send an ACK bit (by clearing the flag and setting the SCTRLB register to a RESPONSE configuration). If the AP flag is not raised, this indicates a stop command was received (after the host received a NACK) and so the SCTRLB register is set to a COMPTRANS configuration to end the transaction and release the clock (which while controlled by the host could still be held by the client in a method known as 'clock stretching'), forcing the bus state to enter an IDLE state. The next interrupt checked is the DIF (Data) which if raised it indicates that there is data ready to be read in the SDATA register (data received from the host).

This implementation of the I^2C protocol is very basic but was still very difficult to develop. It does not handle issues with arbitration, and occasionally the code would get stuck on a blocking while loop that was waiting for either the bus to enter a specific state, or an interrupt flag to be raised. If the bus entered into an Error state, this would not occur and during the initial development of this library, this was indeed the case. While this sufficed for the midterm prototype, seeing this is the lynchpin of the software on each MCU (as without communication the modules would do nothing) this was deemed unfit for purpose for the final product. However, the more I tried to develop this library, the farther from working code I got.

At this point, since time was short due to other aspects of the project (the unsuccessful Sonar Module), I decided to run an experiment utilizing the megaTinyCore[18] Arduino device pack and its version of the popular Wire package[19]. Using Arduino code I developed a simple circuit of interconnected microcontrollers and LEDs where the host device could set the colour of the RGB LEDs on the client and the clients could send a colour for the host (thereby testing both read and write operations). This worked flawlessly, so I decided this would be a good library to use as a basis. However the toolchains I was using (released by microchip) are not capable for compiling C++ code for 8 bit MCUs, so I had to first of all remove references to other libraries (Wire extends Arduino's 'Stream' class) which removed the ability to send characters over this protocol as this allowed for the interpretation of them, and also convert this library to C. I could have written the project in Arduino using this device pack, but as mentioned previously, this

would remove the ability to easily configure the MCU peripherals which I needed the ability to do.

This new implementation (I named CWire since it was the Wire library rewritten in C), while having *some* limitations (only able to send hex or integer values), worked flawlessly and there are a couple of differences between this implementation and my own. The main difference is this version does not use Smart Mode to automatically send ACK or NACK bits, but controls this as necessary in the code by writing to the MCTRLB register. Second, it uses arrays to store data received and to be transmitted so as soon as the DIF flag is raised for example, it is cleared from the appropriate data register almost immediately and temporarily stored until read properly by the host code implementing this library. This aids in freeing up the bus state almost immediately. This library also combines the host and client code which while in this projects design is not necessary as only one device will be designated as host, it means I only have to maintain one library as opposed to two, and this can be exported as a stand alone entity to be used in other future projects using MCUs in the same family. Finally, this code has far more robust error handling for the different bus states, and it also handles arbitration by throwing an error, and cases where the bus is busy it will wait till it is idle before transmitting. The full code for this library can be seen in appendices 1.6

Sonar

The final sonar module incorporated into this project is something I am less than happy with. As mentioned previously, this aspect of the project was something I was hoping to develop from scratch due to the lack of affordable low cost prefabricated options that are designed specifically for underwater use. In the end I had to settle with for a module (JSN-SR04T[32]) that while not rated for underwater use, there are forums that deployed the same sensors underwater had mixed reviews as to how well they worked in this environment. This is a stop gap solution as the only way I will be able to develop this particular module further is outside of the time restrictions imposed by this module. Since this module works in a similar fashion to the HC-SR04 module[7], I was able to use code I had developed in a previous personal project[31]. Despite using similar code to what I had previously used, the rangefinder would not give what I would consider accurate results while testing the device. This could be due to a combination of factors: first, breadboards were used extensively through the development and testing phases of this project. Poor breadboards can sometimes create issues that are not present where a soldered solution is created. This is due to the design of breadboards (metal strips that contact the wires) can produce alot of electrical noise that may prevent some components that are sensitive

from working as expected. This has happened to me before on a previous module integrating LoRa modules. Second, while the HC-SR04 module uses two transducers to transmit and receive, the JSN-SR04T uses a single transducer to perform both operations. In order to get an accurate response the device has to wait for the transducer to stop “ringing” after transmission is ended before it can detect a response, otherwise it could be receiving false positives. This results in an increased blind spot (which is only going to be exacerbated underwater) which could also be impacting the distance calculations. Finally, the transducer is connected to the devices PCB via an extremely long cable which I doubt from the thickness is shielded. This had to be coiled to fit inside the AUV which could have created inductance impacting the sensor. While the results were not accurate, they did appear to scale to the distance from an object, so were able to incorporate this into the final project, with the knowledge that in the future these will hopefully not be apart of the project.

Depth Controller

The depth controller in this AUV is an interesting area. Following similar design principles to The Brick Experiments remote controlled submarine, the depth is controlled using a large syringe drawing in water from outside the AUV to modify the overall mass of the vehicle affecting the bouyancy. The first design challenge was pushing and pulling the syringe. To accomplish this I designed and printed a housing for the syringe which are included in the AUV CAD drawings in appendices 1.7. The housing connects the syringe plunger to a threaded shaft driven by a stepper motor. In order to prevent this from trying to over screw at either end (which could potentially cause physical damage) at either end I created a ‘buffer’. This is a small metal plate with voltage running through it (generated from one of the pins on the MCU to keep it at a safe current). The nut that traverses the thread has two wires attached connected to the MCU so when they touch each end of this ‘buffer’ we can determine that we have reached the end of travel and adjust the position variables in code accordingly. Because these detection wires are floating waiting to detect a voltage I enabled the internal pullup resistors on these pins as without this the pins would often not correctly detect the voltage and the pins have interrupts set to trigger on rising voltage changes. This buffer worked well, but because I did not want the detection pins to potentially sit at a position where it could be continuously drawing power, once detected the position is set to wind itself back to just before the buffer plate is touched. The complete code for the depth controller can be seen in appendices 1.8.10.

The Depth Controller aspect of this project is directly related to the ballast calculations that are necessary to perform to ensure the AUV is

correctly weighted. Bouyant force[34] is the upward force applied to an object against the gravitation downward pull. If the bouyant force is greater than the weight of the object in fluid, the object will float[17]. In order to calculate the required weight to be added to the AUV I have developed a simple python notebook to perform the calculations, which can be seen in the code repository[28].

$$Fb = pVg$$

Using this calculation we can work out the bouyant force applied to the AUV. First, we require the density of the fluid to be displaced (p). While AUVs are typically deployed in the ocean, where salt affects the density we are testing in plain water where the density is known to be approximately 998kg/m^3 . Second we need the volume (V) of the fluid that will be displaced (the volume of the AUV). This is easily obtained from the CAD software and is 0.00762m^3 . Finally we need the gravitational acceleration(g) which is known to be 9.8 . This calculates the Bouyant force to be 74.676N . In order to determine if the AUV will float, we also need to know its weight (in N) which can be simply calculated.

$$N = mass \times g$$

The weight of the AUV is approximately 39.2N . Since this is less than the bouyant force, the AUV without ballast will float. For this AUV to be able to dive and raise effectively without the use of propellers or fins/rudders, it would need to be slightly positively bouyant. This would cause additional water drawn in through the syringe on the depth controller to change the AUV weight so it would fall into negative bouyancy and therefore start to sink. Releasing the water should then cause it to go back to slight positive bouyancy causing the AUV to rise. This is acheived by adding ballast. Since the target bouyancy is 0.1N we need to perform the calculation $b = Fb - W - 0.1$ which is $b = 74.676 - 39.2 - 0.1$. b is therefore the additional weight we need to add as ballast. In order to get the required mass, we simply reverse the mass to newton calculation.

$$mass = \div bg$$

This equation would give us the overall target mass of the AUV which is calculated as $7.61(2\text{d.p})$. To get the mass of ballast required we take off the AUVs current mass which is 4kg . This means we require 3.61kg of additional mass.

Evaluation

(max 3 pages)

The testing and evaluation of this project can be split into two parts: Software/hardware (electronics), and mechanical (meaning the testing of the 3D printed parts and mechanical solutions). While I am an advocate of test driven development, this approach is not the most appropriate for microcontroller programming as the code is very dependent on the hardware, or specifically the state of the bytes in the registers of the hardware. While it would be theoretically possible to mock these states in some kind of simulator, there are some actions that also happen automatically by reading from or writing to certain registers such as interrupt flags being cleared by reading/writing from data registers. Test driven code using mocked simulations that don't account for this could therefore be inaccurate when running the same code on the actual MCU. While unit testing *is* possible for MCU programming, there is no official methodology or libraries provided by Microchip (AVR manufacturer), and forums[6] tend to be of split opinions as to whether this is practical or not owing to the code being tested for simulators not the actual hardware, but with some people stating they do use google test libraries and simulators.

Because of this I chose not to use a TDD approach developing this project. Instead code was manually tested on the devices manually with any unexpected behaviour debugged with the use of the hardware debugger available using the programmer (ATMEL-ICE[26]) programming the device via the Unified Programming and Debug Interface (UPDI) which allows stepping through the code line by line on the Microcontroller itself. This approach is not without its own issues, particularly when trying to debug issues related to data received from other devices over the I^2C protocol as if a device is sent data while the receiver is in a frozen state due to debugging, this could (and did) throw false errors causing the I^2C bus to enter error states. In most cases however this was a very useful tool as it allowed observation of the varying states of the registers at each step of the code execution. When combined with a static code analysis bugs could be traced with relative ease.

The development of the specific software features were manually tested with an attempt to cover as many edge cases as I could think of, but ig-

noring scenarios that were deemed impossible to reach, such as the Depth Controllers buffer hitting the IN and OUT buffers simultaneously is physically impossible. This manual testing method did allow for an iterative approach to the code, where bugs were fixed as observed, but it was time consuming and could not be exhaustive. With more time available it would be worth exploring a TDD approach as there are some aspects of the code that are not reliant on the MCU itself, such as the distance calculations in the Sonar module. Admittedly the testing of the software could have benefited from some consideration prior to starting development by writing some manual test scripts but considering I was testing a debugging simultaneously alongside revising the code, this would have interrupted my workflow.

The mechanical aspects of the project (primarily the 3D printed parts seen in appendices 1.7) were also iteratively designed and tested in the same way as, and often alongside, the software. The depth controller, being a part that had moving parts and needed to operate smoothly was an example of successful iterative testing and designing. The first iteration of this was modelled to house the stepper motor and connect to the threaded shaft but immediately failed its test as this was modelled as a single piece and I did not account for clearance allowing the stepper motor to get inside the housing. Following this design flaw, the part was redesigned to be comprised of individual components that can be put together. This made future adjustments to the model far easier, for example one issue the part had for a while during development was it would get stuck occasionally while turning while the software was trying to perform either a 'raise' or 'dive' operation. While it would move again by itself more often than not, the few seconds it was stuck at a position would throw the timings off on a software level (so where the software *thinks* the plunger is may not be accurate), and it put unnecessary strain on the motor as it required torque to move past this point. The remedy for this was to allow for larger tolerances between the printed parts. By adding an additional 0.2mm clearance on the thread and nut driving the syringe (0.4mm total), the travel was far smoother and mitigated any future requirements for additional components such as grease.

Another mechanical aspect of this project that could have benefitted from earlier testing stages was the mechanism to drive the propellers. This aspect of the project was a bit of a challenge as a hole for the any shaft that provided clearances to allow free movement would also allow the ingress of water. To this end, I decided to attempt a solution using magnets to drive propellers separated by a physical barrier. In this design, the motor with a shaft/gears to step up the speed are on the interior of the vessel, while the propeller itself is on the outside. Connecting these are 2 flat disks perpendicular to the physical wall separating each other but parallel with each other, with equally spaced magnets with alternating polarities. The issue with this design is the motors do not have enough torque to physically turn these disks when they are magnetically attached directly to the wall

seperating them. The only assumption for this is the surface area of the disks offers too much friction for the parts to move freely. On the rear motor this was mitigated by adding a small washer in between reducing the contact area, while still keeping the magnets attracted to each other enough to drive the propeller. The side motors however relied on small toy motors that provide plenty of speed but next to no torque, and while the main propeller was supported by a prop guard, the side motors were designed to be detachable to reduce breakage during travel, and so could not be solved by inserting a washer between and so in this iteration of the final solution, the side propellers will not turn if attached. This is one instance where failure to properly test a solution prior to incorporating it fully into the project was a failure as this could have been identified earlier, and the designed revised before it was too late, while keeping the core concept (magnetically driven) and the inexpensive toy motor in place without having to look at more expensive (in terms of both economically and power consumption) motors that have more torque.

Finally, the waterproofing of this project was one aspect of the mechanical aspects of the project that did go through proper testing and rounds of iterative changes. The test scenario was to print the vehicle, assemble it and submerge it to check for water-tightness. The first test just the AUV sections with only TPU printed gaskets in between and was largely unsuccessful. Water seeped in from every joint, indicated by several streams of bubbles across the AUV. As the sections appeared to be the weak point, the next test was the same again but using silicone sealant to seal these sections. While this test resulted in no bubbles between the sections, water was still getting in to the interior. On closer inspection of the AUV body, there are small holes in the surface of the hull, particularly on the more complex areas such as the side motor pods. This is a negative aspect of using 3D printing to make the hull as with incorrect settings in the slicer, combined with the nature of how these parts are manufactured layer by layer can lead to such imperfections. To mitigate this and to save having to reprint everything at a higher resolution (a process with would have taken in excess of 10 days), each section of the hull was coated in epoxy resin and allowed to cure for over 24 hours. Finally, the vessel was reassembled (a process which was now much harder as the tolerances of the model did not include the thickness of the resin layer) and copious amounts of silicone was applied to every layer. This passed the water tightness test, but as it transpires it only really passed this particular 'dry run' if you'll excuse the pun.

The final test for the AUV was testing the unit as a whole, in a body of water. The initial challenge here was finding a body of water large enough and suitable enough to adequately test. This was found by putting a post on Facebook, in the local community groups to see if anyone had a pool I could borrow for which one individual granted me access to his pool. This test failed. The after placing the AUV in the water, and turning it on, nothing

appeared to happen, and eventually it seemed the bow of the AUV was gradually sitting lower in the water than when it was first put in indicating it was taking on water, which was confirmed by the weight of the AUV when lifting it out. Unfortunately there are too many variables to determine what went wrong in this test. First, because the AUV is a sealed box, I had no visibility on the status LEDs soldered onto the modules to determine if it was operating correctly. It could be that connecting wires came unattached during the assembly process so nothing was working to start. The water ingress, while a problem on this test, highlights that the process to make the vessel water tight is too prone to error and needs further exploration, experimenting with different combinations of materials and printer settings. Another contributing factor is the prop guard at the rear of the AUV was broken in transit which could have provided an avenue for water to seep in. Unfortunately, this test only occurred with a week to go before the final deadline and there is physically not enough time to refine, reprint, and resolder what is required to replace any potentially broken parts in that space. While there are lessons to be learnt from this test (and that is the point of tests), there is not enough time to enact on them in the scope of this module.

Conclusion

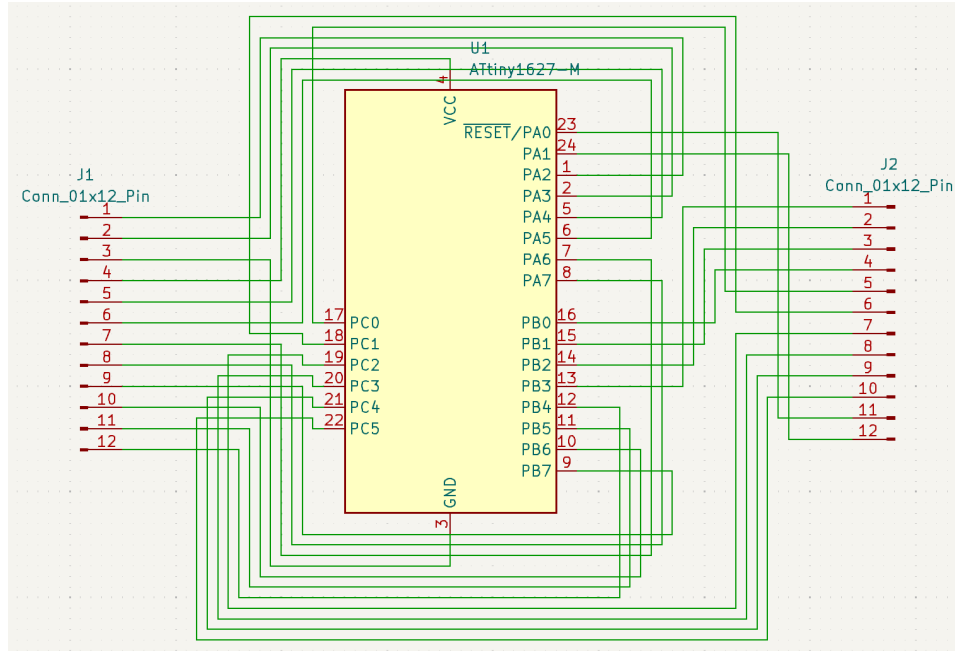
Conclusion: This can be a short summary of the project as a whole but it can also bring out any broader themes you would like to discuss or suggest further work. (max 2 pages)

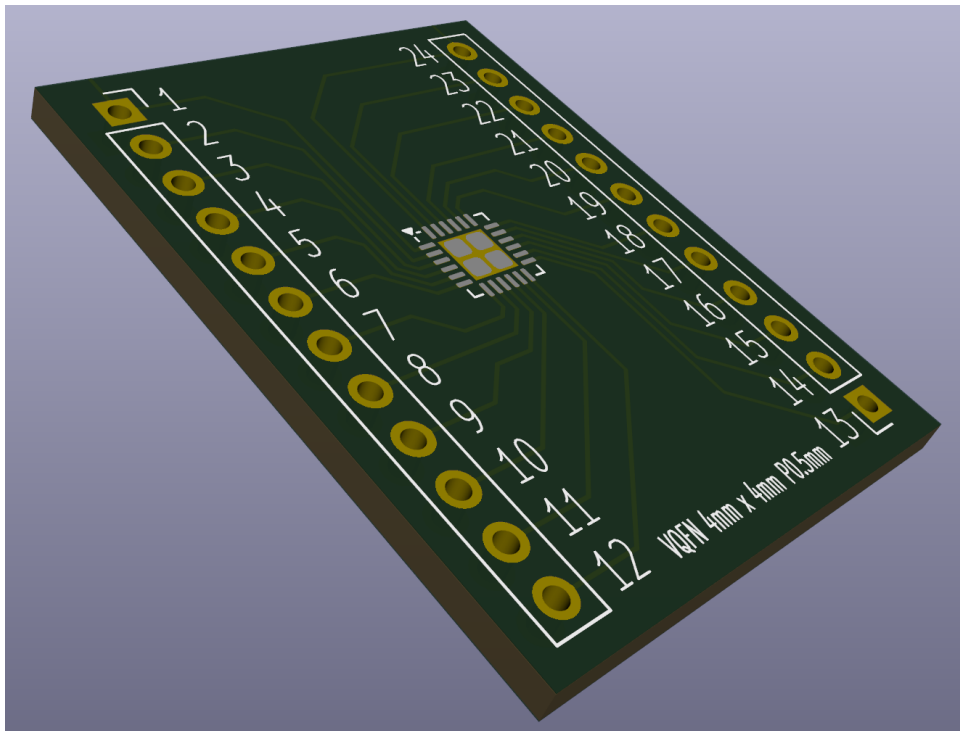
Appendices

1.1 AVR ATTiny 1627 Features

- Low Power
- 16kB programmable flash memory
- 256B EEPROM
- Power on Reset
- Brown out Detection
- Configurable clock
- UPDI
- Idle, Standby and Power down sleep modes
- 3x 16 bit Timer Counters (TCA and 2 x TCB)
- PWM generation
- Real time Counter
- USART
- SPI
- TWI (I^2C)
- Analog to Digital conversion (ADC) with amplifier
- Watchdog Timer
- Interrupts on all GPIO pins
- 22 Programmable GPIO pins

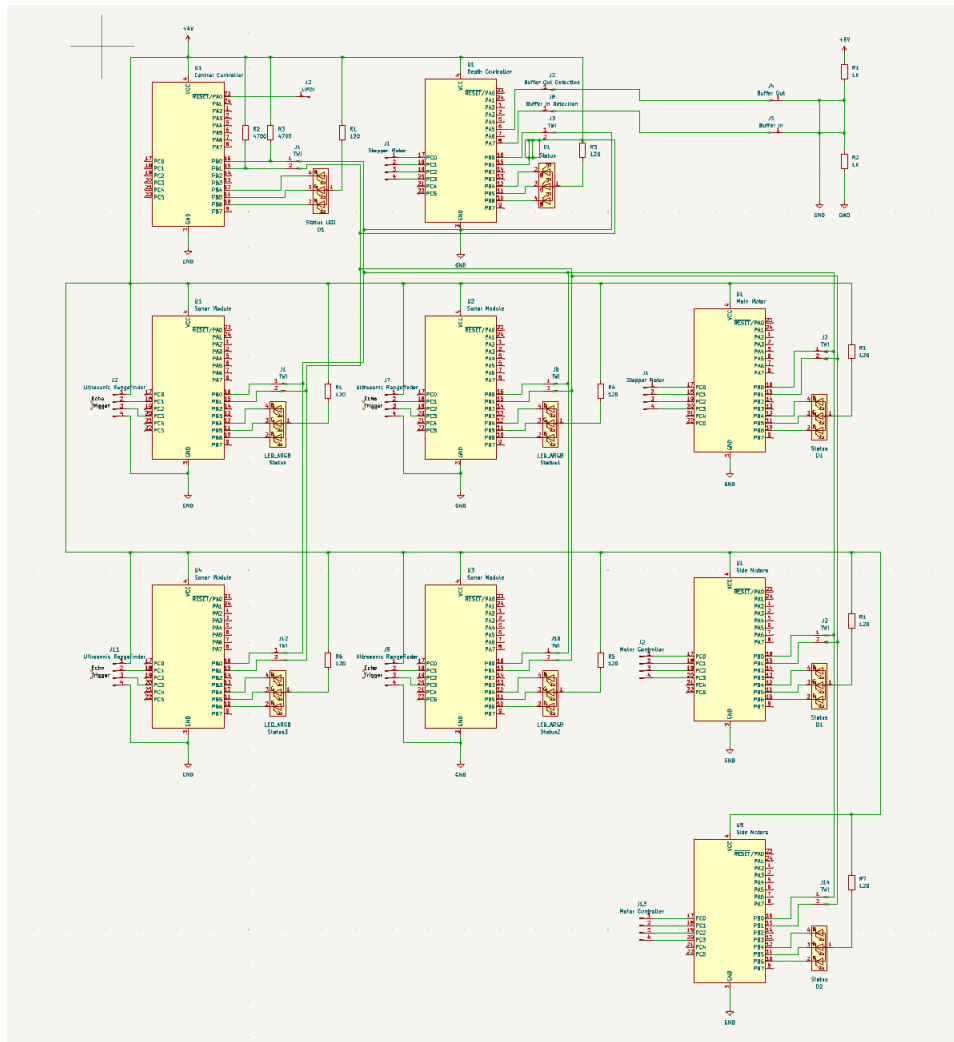
1.2 VQFN Breakout



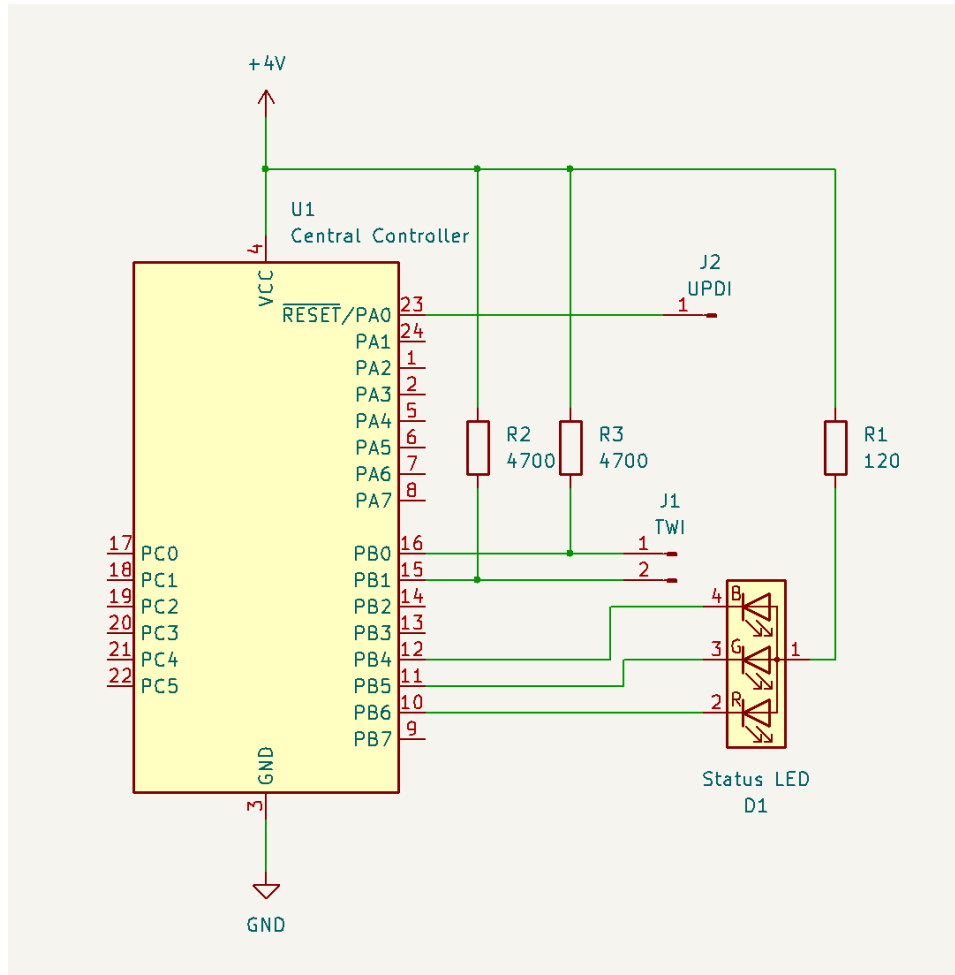


1.3 AUV Electronics Schematics

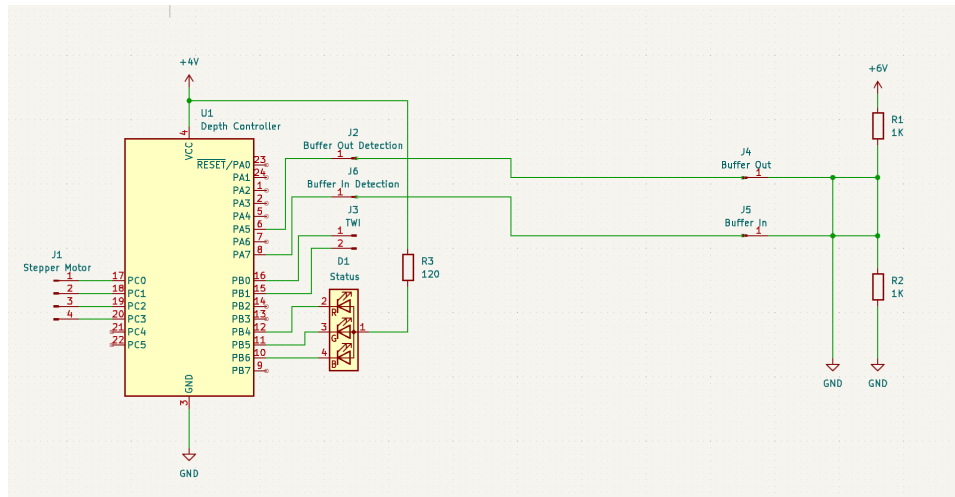
1.3.1 AUV Schematic



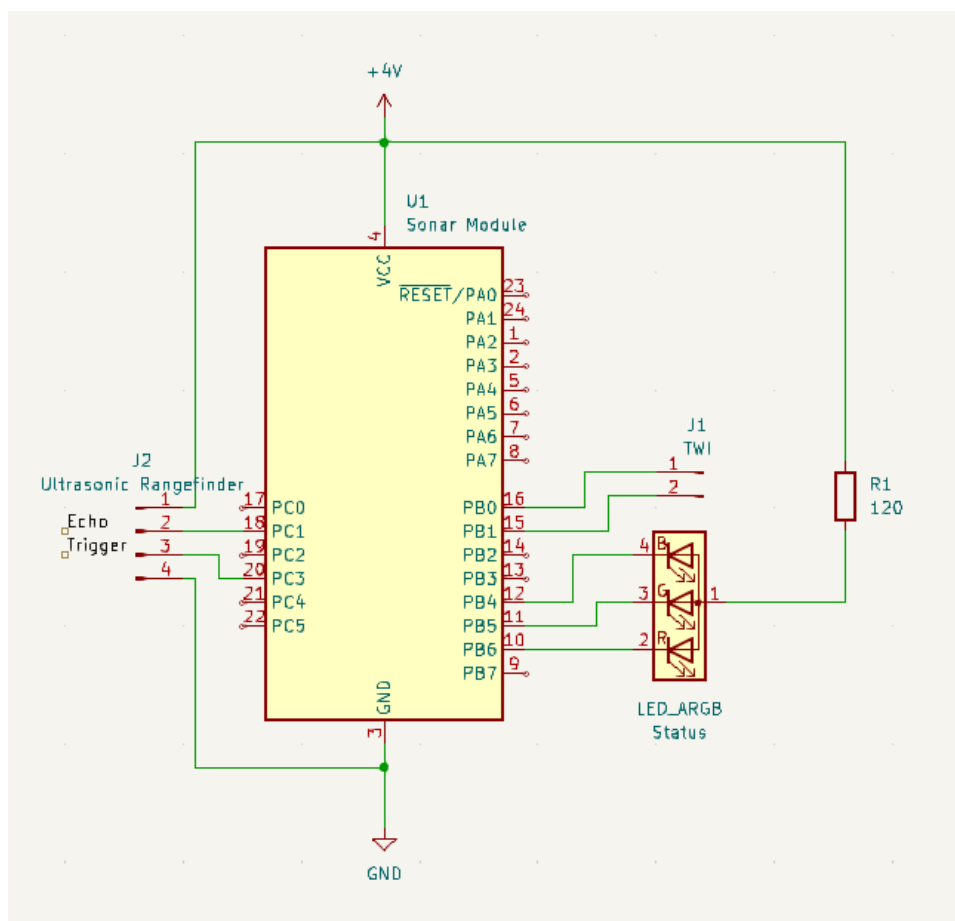
1.3.2 Central Controller Schematic



1.3.3 Depth Controller Schematic

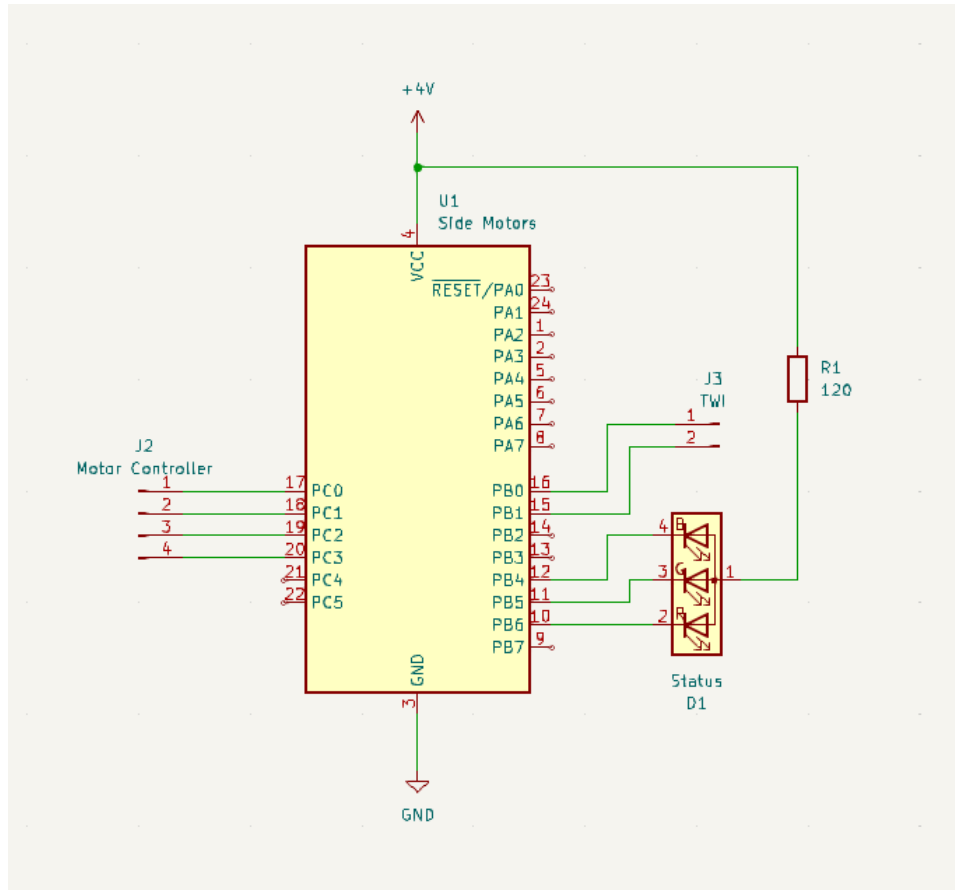


1.3.4 Sonar Module Schematic



The diagram shows a PIC16F877 microcontroller (U1) configured as a stepper motor driver. The PIC is powered by a +4V supply (VCC) and ground (GND). It is connected to a stepper motor (J1) via pins PC0-PC5. The PIC is also connected to a status LED (D1) via pins PB0-PB7. A 120 ohm resistor (R1) is connected to the status LED. The PIC is also connected to a TWI interface (J3) via pins PA0-PA7.

1.3.6 Side Motor Schematic



1.4 Pseudocode

1.4.1 Central Controller

```
1  sonars = [lower, forward, left, right]
2  sonarIndex
3  depth = 255
4
5  function getDepth:
6  return I2C.read(DEPTH_CONTROLLER_ADDR)
7
8  setup:
9  configure RTC for 1 second interrupts
10 configure I2C as Host
11
12 //Need to give the depth controller time to find its home position
```

```

13     while(getDepth() != 1):
14         delay(1000)
15
16     dive(depth)
17
18     //delay to give other modules time to properly initialise
19     delay(2000);
20
21     mainloop:
22     //Does nothing
23
24     function ping(sonar) {
25         switch(sonar):
26         case lower:
27             //write operation to initialise the sonar module
28             I2C.write(LOWER_SONAR_ADDR, PING_COMMAND)
29             //wait for the sonar to finish then collect the results
30             delay(250)
31             return I2C.read(LOWER_SONAR_ADDR)
32         case forward:
33             //write operation to initialise the sonar module
34             I2C.write(FORWARD_SONAR_ADDR, PING_COMMAND)
35             //wait for the sonar to finish then collect the results
36             delay(250)
37             return I2C.read(FORWARD_SONAR_ADDR)
38         case left:
39             //write operation to initialise the sonar module
40             I2C.write(LEFT_SONAR_ADDR, PING_COMMAND)
41             //wait for the sonar to finish then collect the results
42             delay(250)
43             return I2C.read(LEFT_SONAR_ADDR)
44         case right:
45             //write operation to initialise the sonar module
46             I2C.write(RIGHT_SONAR_ADDR, PING_COMMAND)
47             //wait for the sonar to finish then collect the results
48             delay(250)
49             return I2C.read(RIGHT_SONAR_ADDR)
50     }
51
52     function handlePingResponse(sonar, distance):
53         switch(sonar):
54         case lower:
55             if (distance < 10):
56                 depth -= 10
57             else if (distance != 255):
58                 depth += 10
59             if (depth > 255):
60                 depth = 255
61             if (depth < 0):

```

```

62         depth = 0
63         I2C.write(DEPTH_CONTROLLER_ADDR, depth)
64     case forward:
65         if (distance < 10):
66             I2C.write(FORWARD_MOTOR_ADDR, STOP_COMMAND)
67         else
68             I2C.write(FORWARD_MOTOR_ADDR, FORWARD_COMMAND)
69     case left:
70         if (distance < 10):
71             I2C.write(FORWARD_MOTOR_ADDR, STOP_COMMAND)
72             I2C.write(LEFT_MOTOR_ADDR, TURN_COMMAND)
73         else
74             I2C.write(FORWARD_MOTOR_ADDR, FORWARD_COMMAND)
75             I2C.write(LEFT_MOTOR_ADDR, STOP_COMMAND)
76     case right:
77         if (distance < 10):
78             I2C.write(FORWARD_MOTOR_ADDR, STOP_COMMAND)
79             I2C.write(RIGHT_MOTOR_ADDR, TURN_COMMAND)
80         else
81             I2C.write(FORWARD_MOTOR_ADDR, FORWARD_COMMAND)
82             I2C.write(RIGHT_MOTOR_ADDR, STOP_COMMAND)
83
84     //ISRs
85     On_RTC_Interrupt:
86         distance = ping(sonars[sonarIndex])
87         handlePingResponse(sonars[sonarIndex], distance)
88         sonarIndex += 1
89         if (sonarIndex > 3):
90             sonarIndex = 0

```

1.4.2 Depth Controller

```

1     depth = 255
2     commandedPos = 0
3     dir = IN
4
5     setup:
6         configure RTC for 1 second interrupts
7         configure I2C as Client
8         configure Pins for Stepper Motor
9
10    function stepMotor(step):
11        switch(step):
12            case 0:
13                Set pins to 1010
14            case 1:
15                Set pins to 1001

```

```

16         case 2:
17             Set pins to 0101
18         case 3:
19             Set pins to 0110
20         case 4:
21             Set pins to 0011
22         case 5:
23             Set pins to 1100
24         case 6:
25             Set pins to 0100
26         case 7:
27             Set pins to 1000
28
29     function stopMotor():
30         Set pins to 0000
31
32     stepperStep = 0
33     mainloop:
34         if (depth != commandedPos):
35             if (dir == OUT && plungerPos != 255):
36                 stepMotor(stepperStep)
37                 stepperStep -= 1
38                 //There are 8 possible steps (pin configurations) for
39                 the stepper
40                 //motor according to online resources/datasheet*
41                 //*Its a cheap stepper motor so datasheet is a very
42                 loose term.
43                 if (stepperStep < 0):
44                     stepperStep = 7
45
46             if (dir == IN && plungerPos != 0):
47                 stepMotor(stepperStep)
48                 stepperStep += 1
49                 if (stepperStep > 7):
50                     stepperStep = 0
51
52         else:
53             stopMotor()
54
55     //ISRs
56     On_RTC_Interrupt:
57         if (depth != commandedPos):
58             if (depth > commandedPos):
59                 dir = OUT
60                 commandedPos += 1
61             else:
62                 dir = IN
63                 commandedPos -= 1

```

```

63
64     On_I2C_RX:
65         data = I2C.read()
66         if (data > depth):
67             dir = OUT
68         else:
69             dir = IN
70         commandedPos = data
71
72     On_I2C_TX:
73         I2C.write(depth)

```

1.4.3 Main Motor

```

1     dir = FORWARD
2
3     setup:
4         configure I2C as Client
5         configure Pins for Stepper Motor
6
7     function stepMotor(step):
8         switch(step):
9             case 0:
10                 Set pins to 1010
11             case 1:
12                 Set pins to 1001
13             case 2:
14                 Set pins to 0101
15             case 3:
16                 Set pins to 0110
17             case 4:
18                 Set pins to 0011
19             case 5:
20                 Set pins to 1100
21             case 6:
22                 Set pins to 0100
23             case 7:
24                 Set pins to 1000
25
26     function stopMotor():
27         Set pins to 0000
28
29     mainloop:
30         if (dir == FORWARD):
31             stepMotor(stepperStep)
32             stepperStep += 1
33             if (stepperStep > 7):

```

```

34         stepperStep = 0
35     else:
36         stopMotor()
37
38     // ISRs
39     On_I2C_RX:
40         dir = I2C.read()

```

1.4.4 Sonar

```

1     distance = 0
2
3     setup:
4         setup TCA to count to max value
5         configure I2C as Client
6         configure pins for sonar module
7
8     mainloop:
9         //Do nothing. wait for ISR
10
11     function ping():
12         //Pull trigger high and wait 10us
13         TRIGGER_PIN = HIGH
14         delay_us(10)
15         TRIGGER_PIN = LOW
16         TCA = ENABLE
17         while (ECHO_PIN == HIGH):
18             //Wait for echo pin to go low
19
20         ticks = TCA.COUNT
21
22         //Convert the TCA ticks to a time in uS
23         clk = F_CPU/ 64
24         time = ticks / clk
25
26         sos = 148000
27         //This should give us the distance in cm/uS
28         distance = (time * sos) / 2
29
30     // ISRs
31     On_I2C_RX:
32         distance = ping()
33
34     On_I2C_TX:
35         I2C.write(distance)

```

1.4.5 Side Motor

```
1      setup:
2          configure Pins for toy motor
3          configure I2C as Client
4
5      mainloop:
6          //Do nothing. wait for ISR
7
8      // ISRs
9      On_I2C_RX:
10         dir = I2C.read()
11         if (dir == FORWARD):
12             //Turn the motor on
13             //twin motor driver.
14             //00 = off
15             //01 = forward
16             //10 = reverse
17             //11 = brake
18             MOTOR_PINS = 1010
19         else:
20             //Turn the motor off
21             MOTOR_PIN = 0000
```

1.5 Midterm I^2C

1.5.1 host.h

```
1      #ifndef I2C_HOST_H
2      #define I2C_HOST_H
3
4      #ifndef F_CPU
5      #define F_CPU 3333333UL
6      #endif
7      #include <avr/io.h>
8
9      #define I2C_CHECK_START() (TWI_WIF_bm & TWI_CLKHOLD_bm)
10     #define I2C_CHECK_WRITE() (TWI_WIF_bm)
11     #define I2C_CHECK_NACK() (TWI_WIF_bm & TWI_RXACK_bm)
12
13     typedef struct {
14         // Add any necessary fields here, if needed
15     } I2C_Host;
16
17     void I2C_Host_InitPins(void);
18
19     void I2C_Host_InitI2C(void);
```



```

20
21     uint8_t I2C_Host_CalcBaud(void);
22
23     struct BAUD_TIMING_t{
24         double T_LOW;
25         double T_HIGH;
26         double T_OF;
27         double T_R;
28     };
29
30     double I2C_Host_CalcBaud_F_SCL(void);
31
32     double I2C_Host_CalcBaud_BAUD(double);
33
34     uint8_t I2C_Host_Start(uint8_t, uint8_t);
35
36     void I2C_Host_WriteData(uint8_t);
37
38     void I2C_Host_Stop(void);
39
40     #endif // I2C_HOST_H

```

1.5.2 host.c

```

1
2     #include "I2C_Host.h"
3
4     void I2C_Host_InitPins(void)
5     {
6         /**
7          * Data sheet claims 1627 pins are the following:
8          *
9          * PB0: SCL
10         * PB1: SDA
11         */
12
13         PORTB.DIR |= PIN0_bm | PIN1_bm;
14
15         PORTB.PINCTRL |= PORT_PULLUPEN_bm;
16         PORTB.PIN1CTRL |= PORT_PULLUPEN_bm;
17     }
18
19     void I2C_Host_InitI2C(void)
20     {
21         //Set the Master Baud Rate (Master defines baud for clients)
22         // TWI0.MBAUD = I2C_Host_CalcBaud();
23         TWI0.MBAUD = 12;

```

```

24
25      //Set the bus state to IDLE
26      TWIO.MSTATUS |= TWI_BUSSTATE_IDLE_gc;
27
28      //Enable Smart Mode. We need to set MCTRLB ACKACT set to ACK
29      //Then MCTRLA SMEN (Smart Mode Enable) set to 1
30      TWIO.MCTRLB = TWI_ACKACT_ACK_gc;
31
32      //Enable the TWI
33      TWIO.MCTRLA = TWI_SMEN_bm | TWI_ENABLE_bm;
34  }
35
36  uint8_t I2C_Host_CalcBaud(void)
37  {
38      double fSCL = I2C_Host_CalcBaud_F_SCL();
39      double baud = I2C_Host_CalcBaud_BAUD(fSCL);
40      return (uint8_t)(baud);
41  }
42
43
44  struct BAUD_TIMING_t BAUD_TIMING = {
45      .T_LOW = 5.0e-6,
46      .T_HIGH = 5.0e-6,
47      .T_OF = 0,
48      .T_R = 0
49  }; //These values should produce time of 100000
50
51  double I2C_Host_CalcBaud_F_SCL(void)
52  {
53      //f(SCL) = 1 / t_LOW + t_HIGH + t_OF + t_R
54      /**
55       * From the timing requirements we can infer that:
56       *
57       * t_LOW:                MIN VALUE                MAX VALUE
58       * @ <= 100kHz           4.7us
59       * @ <= 400kHz           1.3us
60       * @ <= 1MHz             0.5us
61       *
62       * t_HIGH
63       * @ <= 100kHz           4.0us
64       * @ <= 400kHz           0.6us
65       * @ <= 1MHz             0.26us
66       *
67       * t_OF
68       * @ <= 100kHz           -                        250ns
69       * @ <= 400kHz           20*(Vdd / 5.5)ns          250ns
70       * @ <= 1MHz             20*(Vdd / 5.5)ns          120ns
71       *
72       * t_R

```

```

73      *      @ <= 100kHz      -      1000ns
74      *      @ <= 400kHz      20ns      300ns
75      *      @ <= 1MHz      -      120ns
76      */
77      //init values in seconds
78
79      double total = 5.0e-6 + 5.0e-6 + 0 + 0;
80      double fSCL = 1.0 / total;
81      return fSCL;
82  }
83
84  double I2C_Host_CalcBaud_BAUD(double fScl)
85  {
86      double baud = (F_CPU / 2*(fScl)) - (5 + ((F_CPU *
87          BAUD_TIMING.T_R) / 2));
88      return baud;
89  }
90
91  uint8_t I2C_Host_Start(uint8_t addr, uint8_t dir)
92  {
93      while ((TWIO.MSTATUS & TWI_BUSSTATE_gm) !=
94          TWI_BUSSTATE_IDLE_gc);
95      //Write the address shifting one to the left. dir bit
96      //determines
97      //if Read(0x00) or write(0x01) operation
98      TWIO.MADDR |= (addr << 1) | dir;
99
100     //We only want to check the flags are high on write mode
101     if (dir == 0x00)
102     {
103         //Check Interrupt Flags are high
104         //      while(!(TWIO.MSTATUS & I2C_CHECK_START()));
105         while (!(TWIO.MSTATUS & TWI_WIF_bm) || !(TWIO.MSTATUS &
106             TWI_CLKHOLD_bm));
107     }
108
109     // all is good, return truthy
110     return 0x01;
111 }
112
113 void I2C_Host_WriteData(uint8_t data)
114 {
115     //Writing to MData clears the WIF (Write Interrupt Flag)
116     TWIO.MDATA = data;
117
118     //Check for WIF
119     while(!(TWIO.MSTATUS & I2C_CHECK_WRITE()))
120     {
121         if (TWIO.MSTATUS & I2C_CHECK_NACK())

```

```

118         {
119             //Error Handle. For now call stop?
120             I2C_Host_Stop();
121             return;
122         }
123     }
124 }
125
126 void I2C_Host_Stop(void)
127 {
128     //Send the Stop Command on MCTRLB
129     TWIO.MCTRLB |= TWI_MCMD_STOP_gc;
130 }

```

1.5.3 client.h

```

1     #ifndef I2C_CLIENT_H
2     #define I2C_CLIENT_H
3
4     #ifndef F_CPU
5     #define F_CPU 3333333UL
6     #endif
7     #include <avr/io.h>
8     #include <stdint.h> // Include stddef.h for NULL
9     #include <avr/interrupt.h>
10
11     // Type definition for the receive callback function
12     typedef void (*I2C_ReceiveCallback)(uint8_t data);
13
14     // Function declarations
15     void I2C_Client_InitPins(void);
16     void I2C_Client_InitI2C(uint8_t address, I2C_ReceiveCallback
17         callback);
18     uint8_t I2C_Client_ReadData(void);
19     void I2C_Client_WriteData(uint8_t data);
20     #endif // I2C_CLIENT_H

```

1.5.4 client.c

```

1     #include "I2C_Client.h"
2
3     // Global variable to store the receive callback function
4     static I2C_ReceiveCallback receive_callback = NULL;
5
6     // Initialize I2C pins

```

```

7 void I2C_Client_InitPins(void)
8 {
9     // Set PB0 (SCL) and PB1 (SDA) as input
10    PORTB.DIR &= ~(PIN0_bm | PIN1_bm);
11 }
12
13 // Initialize I2C in client mode
14 void I2C_Client_InitI2C(uint8_t address, I2C_ReceiveCallback
    callback)
15 {
16     // Store the callback function
17     receive_callback = callback;
18
19     // Set the slave address
20     TWIO.SADDR = address << 1 | 0x00; ///0x01 respond to all
        addr.
21
22     // Enable the TWI and Smart Mode, clear collision and bus
        error flags
23     TWIO.SCTRLA = TWI_ENABLE_bm | TWI_DIEN_bm | TWI_APIEN_bm |
        TWI_PIEN_bm;
24
25     // Ensure the slave interface is enabled
26     // TWIO.SCTRLB = 0;
27     sei();
28 }
29
30 // Read data from I2C
31 uint8_t I2C_Client_ReadData(void)
32 {
33     // Wait for data to be received
34     while (!(TWIO.SSTATUS & TWI_DIF_bm));
35
36     // Check if data was received
37     if (TWIO.SSTATUS & TWI_DIF_bm)
38     {
39         // Read and return the data
40         return TWIO.SDATA;
41     }
42
43     // Return 0 if no data was received
44     return 0;
45 }
46
47 // Write data to I2C
48 void I2C_Client_WriteData(uint8_t data)
49 {
50     // Wait for the data register to be empty
51     while (!(TWIO.SSTATUS & TWI_DIF_bm));

```

```

52
53     // Write the data
54     TWIO.SDATA = data;
55
56     // Wait for the data to be transmitted
57     while (!(TWIO.SSTATUS & TWI_DIF_bm));
58 }
59
60 // Interrupt Service Routine for I2C
61 ISR(TWIO_TWIS_vect)
62 {
63     // Check if Address/Stop interrupt
64     if (TWIO.SSTATUS & TWI_APIF_bm)
65     {
66         // Clear the interrupt flag
67         TWIO.SSTATUS = TWI_APIF_bm;
68
69         // Check if the address match
70         if (TWIO.SSTATUS & TWI_AP_bm)
71         {
72             // Clear the address match flag
73             TWIO.SSTATUS = TWI_AP_bm;
74             TWIO.SCTRLB = TWI_SCMD_RESPONSE_gc;
75         }
76         else
77         {
78             // If not an address match, it must be a stop condition
79             TWIO.SCTRLB = TWI_SCMD_COMPTRANS_gc;
80         }
81     }
82
83     // Check if Data interrupt
84     if (TWIO.SSTATUS & TWI_DIF_bm)
85     {
86         // Read the received data
87         uint8_t received_data = TWIO.SDATA;
88
89         // Call the receive callback if it's set
90         if (receive_callback != NULL)
91         {
92             receive_callback(received_data);
93         }
94
95         // Clear the data interrupt flag
96         TWIO.SSTATUS = TWI_DIF_bm;
97     }
98 }

```

Bibliography

- [1] avrdude. Avrdude documentation. 2023. URL: <https://avrdudes.github.io/avrdude/7.2/avrdude.pdf>.
- [2] I. Bus. Fast mode. URL: <https://www.i2c-bus.org/fastmode/>.
- [3] B. E. Channel. Brick experiment channel. URL: <https://www.youtube.com/c/BrickExperimentChannel>.
- [4] B. E. Channel. Brick experiment channel. URL: <https://brickexperimentchannel.wordpress.com/>.
- [5] ecoSub. Ecosub. URL: <https://www.ecosub.uk/>.
- [6] A. Freaks. Unit test in microchip studio. URL: <https://www.avrfreaks.net/s/topic/a5C31000000UcMhEAK/t161267>.
- [7] E. Freaks. Ultrasonic ranging module hc - sr04. URL: <https://cdn.robotshop.com/media/s/spa/rb-spa-1388/pdf/hcsr04-ultrasonic-range-finder.pdf>.
- [8] gcc. Avr-gcc documentation. 2024. URL: <https://gcc.gnu.org/wiki/avr-gcc>.
- [9] Z. Hong, X. Pan, P. Chen, X. Su, N. Wang, and W. Lu. A topology control with energy balance in underwater wireless sensor networks for iot-based applications. *Sensors*, 2018.
- [10] W. HR. Spurv - the first decade, 1973.
- [11] F. B. Insights. Autonomous underwater vehicle (auv) market size, share & covid-19 impact analysis, by type (small, medium, and large), by application (scientific, research , defense, and oil & gas industry), by propulsion system (electric system, mechanical system, and hybrid system), by payload (camera, sensors, inertial navigation system, and others), and regional forecast, 2021-2028. URL: <https://www.fortunebusinessinsights.com/segmentation/autonomous-underwater-vehicle-market-105907>.
- [12] N. O. Institute. Autosubs. URL: <https://noc.ac.uk/facilities/marine-autonomous-robotic-systems/autosubs>.
- [13] W. H. O. Institute. Auvs. URL: <https://www.who.edu/what-we-do/explore/underwater-vehicles/auvs/>.

- [14] W. H. O. Institute. Remus. URL: <https://www.whoi.edu/what-we-do/explore/underwater-vehicles/auvs/remus/>.
- [15] W. H. O. Institute. Spray glider. URL: <https://www.whoi.edu/what-we-do/explore/underwater-vehicles/auvs/spray-glider/>.
- [16] KiCad. Kicad. URL: <https://www.kicad.org/>.
- [17] L. Learning. Archimedes principle and bouyancy. URL: <https://courses.lumenlearning.com/suny-osuniversityphysics/chapter/14-4-archimedes-principle-and-buoyancy/>.
- [18] megaTinyCore. Spence konde. URL: <https://github.com/SpenceKonde/megaTinyCore>.
- [19] megaTinyCore TwoWire. Spence konde. URL: <https://github.com/SpenceKonde/megaTinyCore/tree/master/megaavr/libraries/Wire/src>.
- [20] Microchip. Attiny1624/1626/1627 data sheet. URL: <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/ATtiny1624-26-27-DataSheet-DS40002234B.pdf>.
- [21] A. Navigation. Hydrus. URL: <https://www.advancednavigation.com/robotics/micro-auv/hydrus>.
- [22] B. News. Remaining debris from destroyed titan sub found on atlantic seabed. Oct. 11, 2024. URL: <https://www.bbc.co.uk/news/world-us-canada-67073535>.
- [23] P. Oceanography. Underwater robots: autonomous underwater vehicles (auvs), 1999.
- [24] U. of London. Module author interviews - project templates. URL: [NEEDED](#).
- [25] Ofcom. Uk frequency allocation table. Aug. 2022. URL: <http://static.ofcom.org.uk/static/spectrum/fat.html>.
- [26] RS-Online. Atmel-ice. URL: https://uk.rs-online.com/web/p/chip-programmers/1306122?cm_mmc=UK-PPC-DS3A-_-google-_-3_UK_EN_Semiconductors_Chip+Programmers-_-Microchip+-+1306122+-+ATAMEL-ICE-_-atatmel+ice&matchtype=e&kwd=66984139136&cq_src=google_ads&cq_cmp=12689163037&cq_term=atatmel%20ice&cq_plac=&cq_net=g&cq_plt=gp&gad_source=1&gclid=CjwKCAjwreW2BhBhEiwAavLwfFbf23eDzecxRUQwEWtZWzJqNBjkdBgyD1kSa7o01rrOHBVbBwE&gclsrc=aw.ds.
- [27] M. Pro. Ultrasonic sensor. URL: <https://www.farnell.com/datasheets/3771814.pdf>.
- [28] RichardSefton. Auv repository. URL: <https://github.com/RichardSefton/AUV>.

- [29] B. Robotics. Ping2 sonar altimeter and echosounder. URL: <https://bluerobotics.com/store/sonars/echosounders/ping-sonar-r2-rp/>.
- [30] R. Sefton. Project template. URL: [NEEDED](#).
- [31] R. Sefton. Stuart. URL: <https://github.com/RichardSefton/STUART/tree/master/STUART.X>.
- [32] P. Supplies. Jsn-sr04t v3.0 waterproof ultrasonic range finder. URL: <https://protosupplies.com/product/jsn-sr04t-v3-0-waterproof-ultrasonic-range-finder/>.
- [33] P. Visconti, G. Giannotta, R. Brama, P. Primiceri, A. Malvasi, and A. Centuori. Feature, operation principle and limits of spi and i2c communication protocols for smart objects: a novel spi-based hybrid protocol especially suitable for iot applications. *International Journal on Smart Sensing and Intelligent Systems*, 10:262–295, 2017.
- [34] P. Walker and K. Wordeman. How to calculate the bouyant force of a floating object. URL: <https://study.com/skill/learn/how-to-calculate-the-buoyant-force-of-a-floating-object-explanation.html>.
- [35] E. Williams. *Make: AVR Programming*. Maker Media Inc., 2015.
- [36] R. B. Wynn, V. A. Huvenne, T. P. L. Bas, B. J. Murton, D. P. Connolly, B. J. Bett, H. A. Ruhl, K. J. Morris, J. Peakall, D. R. Parsons, E. J. Summer, S. E. Darby, R. M. Dorrell, and J. E. Hunt. Autonomous underwater vehicles (auvs): their past, present and future contributions to the advancement of marine geoscience. *Elsevier*, 352:451–468, 2014.

1.6 CWire Library

1.6.1 CWire.h

```

1  #ifndef CWIRE_H
2  #define CWIRE_H
3
4  #include<stdint.h>
5  #include <avr/io.h>
6  #include "Common.h"
7
8  /* The Wire library unfortunately needs TWO buffers, one for
   TX, and one for RX. That means, multiply these
9  * values by 2 to get the actual amount of RAM they take. You
   can see that on the smallest ram sizes, all but
10 * the minuscule buffer we provide becomes prohibitive. 32b is
   a magic number because it's used on the stock

```

```

11      * uno/etc cores, and many libraries rely on it implicitly.
12      * / System RAM / Buffer Size / Applies to
13      *
14      * / Under 256b /      2x16b / tinyAVR 0/1-series with 2k of
15      * / 256-4095b /      2x32b / All other tinyAVR 0/1/2-series,
16      * /   >= 4096b /      2x130b / Dx, EA, and megaAVR 0-series
17      * / with 32k flash or more      /
18      *
19      * On parts that get the reduced buffer size to fit in their
20      * limited RAM, the flash is also very small,
21      * and while the enhanced wire library will fit on 2k parts,
22      * you have very little flash left for anything else.
23      * and the practicality of using it there is limited.
24      */
25
26      #ifndef ADD_READ_BIT
27      #define ADD_READ_BIT(address) (address | 0x01)
28      #endif
29
30      #ifndef ADD_WRITE_BIT
31      #define ADD_WRITE_BIT(address) (address & ~0x01)
32      #endif
33
34      #ifndef DEFAULT_FREQUENCY
35      #define DEFAULT_FREQUENCY 100000
36      #endif
37
38      #if (!defined(TWI1) && defined(TWI_USING_WIRE1))
39      // If pins for Wire1 are not defined, but TWI_USING_WIRE1 was
40      // defined in the boards.txt menu, throw an error. Used for
41      // 28-pin DA/DB parts
42      #error "This part only provides a single Wire interface."
43      #endif
44
45      /* Instead of requiring changes to the library to switch
46      * between DxCore and megaTinyCore, we can check
47      * if the part supports dual mode. Goal is that the identical
48      * library can be used on both, so updates
49      * in one can be propagated to the other by just copying files.
50      */
51
52      #if ((defined(TWIO_DUALCTRL) && !defined(TWI_USING_WIRE1)) ||
53          (defined(TWI1_DUALCTRL) && defined(TWI_USING_WIRE1)))
54      #define TWI_DUALCTRL // This identifies if the device supports
55      dual mode, where slave pins are different from the master
56      pins

```

```

47     #endif
48
49
50     #if defined(__AVR_ATtiny202__) || defined(__AVR_ATtiny202__)
51     #if defined(TWI_MANDS) // 202 and 402 do not support
52         independent master and slave.
53     // #undef TWI_MANDS
54     #error "Master + Slave mode is not supported on the 202 or
55         402."
56     // If a user enables master + slave mode on a part where we
57     know it won't we should error
58     // so that they know what's wrong instead of silently
59     disobeying
60     #endif
61     #endif
62
63     // WIRE_HAS_END means Wire has end(), which almost all
64     implementations do.
65     #ifndef WIRE_HAS_END
66     #define WIRE_HAS_END 1
67     #endif
68
69     // These can be used to write clearer code when using the
70     three-argument begin()
71     // An alternate address is specified by leftshifting and
72     setting the low bit to 1
73     #define WIRE_ALT_ADDRESS(addr) ((addr << 1) | 0x01)
74     // While a mask is specified by leftshifting the mask, and
75     leaving low bit 0.
76     #define WIRE_ADDRESS_MASK(mask) (mask << 1)
77
78     #define WIRE_SDA_HOLD_OFF 0
79     #define WIRE_SDA_HOLD_50 1
80     #define WIRE_SDA_HOLD_300 2
81     #define WIRE_SDA_HOLD_500 3
82
83     #define WIRE_SDA_SETUP_4 0
84     #define WIRE_SDA_SETUP_8 1
85
86     #define WIRE_I2C_LEVELS 0
87     #define WIRE_SMBUS_LEVELS 1
88
89     #if !defined(TWI_BUFFER_LENGTH) // we need a Buffer size
90     #if defined(BUFFER_LENGTH) && (BUFFER_LENGTH != 32) //
91         Backwards Compatibility: someone needs a non-default size
92     #define TWI_BUFFER_LENGTH BUFFER_LENGTH // Apply it.

```

```

87      #warning "It seems like BUFFER_LENGTH was used to change the
        default Wire Buffer Size."
88      #warning "The define was renamed to TWI_BUFFER_LENGTH to
        reduce ambiguity."
89      #warning "TWI_BUFFER_LENGTH was set to BUFFER_LENGTH." //
        defining TWI_BUFFER_LENGTH= will remove the warnings
90
91      #else // BUFFER_LENGTH was not
        messed with, go with our defaults
92      #if (RAMSIZE < 256)
93      #define TWI_BUFFER_LENGTH 16
94      #elif (RAMSIZE < 4096)
95      #define TWI_BUFFER_LENGTH 32
96      #else
97      #define TWI_BUFFER_LENGTH 130
98      #endif
99      #endif
100     #endif
101
102     // In the case someone wants to use custom 255+ buffer sizes,
        define TWI_16BIT_BUFFER
103     #if (TWI_BUFFER_LENGTH > 255) || defined(TWI_16BIT_BUFFER)
104     typedef uint16_t twi_buf_index_t;
105     #else
106     typedef uint8_t twi_buf_index_t;
107     #endif
108
109
110     #define TWI_TIMEOUT_ENABLE // Enabled by default, might be
        disabled for debugging or other reasons
111     #define TWI_ERROR_ENABLED // Enabled by default, TWI
        Master Write error functionality
112     #define TWI_READ_ERROR_ENABLED // Enabled on Master Read too
113     // #define DISABLE_NEW_ERRORS // Disables the new error codes
        and returns TWI_ERR_UNDEFINED instead.
114
115     // Errors from Arduino documentation:
116     #define TWI_ERR_SUCCESS 0x00 // Default
117     #define TWI_ERR_DATA_TOO_LONG 0x01 // Not used here; data too
        long to fit in TX buffer
118     #define TWI_ERR_ACK_ADR 0x02 // Address was NACKed on
        Master write
119     #define TWI_ERR_ACK_DAT 0x03 // Data was NACKed on
        Master write
120     #define TWI_ERR_UNDEFINED 0x04 // Software can't tell
        error source
121     #define TWI_ERR_TIMEOUT 0x05 // TWI Timed out on data
        rx/tx
122

```

```

123      // Errors that are made to help finding errors on TWI lines.
124      Only here to give a suggestion of where to look - these
125      may not always be reported accurately.
126      #if !defined(DISABLE_NEW_ERRORS)
127      #define TWI_ERR_UNINIT      0x10 // TWI was in bad state when
128      function was called.
129      #define TWI_ERR_PULLUP      0x11 // Likely problem with
130      pull-ups
131      #define TWI_ERR_BUS_ARB      0x12 // Bus error and/or
132      Arbitration lost
133      #define TWI_ERR_BUF_OVERFLOW 0x13 // Buffer overflow on master
134      read
135      #define TWI_ERR_CLKHLD      0x14 // Something's holding the
136      clock
137      #else
138      // DISABLE_NEW_ERRORS can be used to more completely emulate
139      the old error reporting behavior; this should rarely be
140      needed.
141      #define TWI_ERR_UNINIT      TWI_ERR_UNDEFINED // TWI was in
142      bad state when method was called.
143      #define TWI_ERR_PULLUP      TWI_ERR_UNDEFINED // Likely
144      problem with pull-ups
145      #define TWI_ERR_BUS_ARB      TWI_ERR_UNDEFINED // Bus error
146      and/or Arbitration lost
147      #define TWI_ERR_BUF_OVERFLOW TWI_ERR_UNDEFINED // Buffer
148      overflow on master read
149      #define TWI_ERR_CLKHLD      TWI_ERR_UNDEFINED // Something's
150      holding the clock
151      #endif
152
153      #if defined(TWI_ERROR_ENABLED)
154      #define TWI_ERROR_VAR      twi_error
155      #define TWI_INIT_ERROR      uint8_t TWI_ERROR_VAR = TWI_ERR_SUCCESS
156      #define TWI_GET_ERROR      TWI_ERROR_VAR
157      #define TWI_CHK_ERROR(x)    TWI_ERROR_VAR == x
158      #define TWI_SET_ERROR(x)    TWI_ERROR_VAR = x
159      #else
160      #define TWI_ERROR_VAR      {}
161      #define TWI_INIT_ERROR      {}
162      #define TWI_GET_ERROR      {0}
163      #define TWI_CHK_ERROR(x)    (true)
164      #define TWI_SET_ERROR(x)    {}
165      #endif
166
167      #if defined(TWI_READ_ERROR_ENABLED) &&
168      defined(TWI_ERROR_ENABLED)
169      #define TWIR_ERROR_VAR      twir_error
170      #define TWIR_INIT_ERROR      uint8_t TWIR_ERROR_VAR =
171      TWI_ERR_SUCCESS

```

```

156     #define TWIR_GET_ERROR      TWIR_ERROR_VAR
157     #define TWIR_CHK_ERROR(x)   TWIR_ERROR_VAR == x
158     #define TWIR_SET_ERROR(x)   TWIR_ERROR_VAR = x
159
160     // #define TWI_SET_EXT_ERROR(x) TWI_ERROR_VAR = x
161     #else
162     #define TWIR_ERROR_VAR      {}
163     #define TWIR_INIT_ERROR     {}
164     #define TWIR_GET_ERROR      {0}
165     #define TWIR_CHK_ERROR(x)   (true)
166     #define TWIR_SET_ERROR(x)   {}
167
168     // #define TWI_SET_EXT_ERROR(x) {}
169     #endif
170
171     typedef struct {
172         u8 _toggleStreamFn;
173         u8 _hostEnabled;
174         u8 _clientEnabled;
175         u8 _hostDataSent;
176         u8 _reserved;
177     } twiDataBools;
178
179     typedef struct {
180         TWI_t *_module;
181         u8 client_irq_mask;
182         u8 _hostBuffer[TWI_BUFFER_LENGTH];
183         u8 _clientBuffer[TWI_BUFFER_LENGTH];
184         u8 _clientAddress;
185         twi_buf_index_t _bytesToReadWrite;
186         twi_buf_index_t _bytesReadWritten;
187         twi_buf_index_t _bytesTransmittedS;
188         twiDataBools _bools;
189         void (*user_onReceive)(u8);
190         void (*user_onRequest)(void);
191     } TwoWire;
192
193     #if (TWI_BUFFER_LENGTH > 255) || defined(TWI_16BIT_BUFFER)
194     typedef uint16_t twi_buf_index_t;
195     #else
196     typedef uint8_t twi_buf_index_t;
197     #endif
198
199     u8 MasterCalcBaud(u32 freq);
200
201     void TwoWire_init(TwoWire *self, TWI_t *twi_module);
202     u8 TwoWire_pins(TwoWire *self, u8 sda, u8 scl);
203     //u8 TwoWire_swap(TwoWire *self, u8 state);
204     //u8 TwoWire_swapModule(TwoWire *self, TWI_t *twi_module);

```

```

205 void TwoWire_usePullups(TwoWire *self);
206 u8 TwoWire_setClock(TwoWire *self, u32 freq);
207
208 // Master
209 void TwoWire_Master_begin(TwoWire *self);
210 void TwoWire_endMaster(TwoWire *self);
211 void TwoWire_beginTransmission(TwoWire *self, u8 addr);
212 u8 TwoWire_endTransmission(TwoWire *self, u8 sendStop);
213 twi_buf_index_t TwoWire_requestFrom(TwoWire *self, u8 addr,
    twi_buf_index_t qty, u8 sendStop);
214 u8 TwoWire_masterTransmit(TwoWire *self, twi_buf_index_t
    *length, u8 *buff, u8 addr, u8 sendStop);
215 u8 TwoWire_masterReceive(TwoWire *self, u16 *length, u8 *buff,
    u8 addr, u8 sendStop);
216
217 // Slave
218 void TwoWire_Slave_begin(TwoWire *self, u8 addr, u8 bc, u8
    sec_addr);
219 void TwoWire_endSlave(TwoWire *self);
220 u8 TwoWire_getIncomingAddress(TwoWire *self);
221 twi_buf_index_t TwoWire_getBytesRead(TwoWire *self);
222 u8 TwoWire_slaveTransactionOpen(TwoWire *self);
223 u8 TwoWire_checkPinLevels(TwoWire *self);
224 //void TwoWire_selectSlaveBuffer(TwoWire *self);
225 //void TwoWire_deselectSlaveBuffer(TwoWire *self);
226 void TwoWire_HandleSlaveIRQ(TwoWire *wire_s);
227
228 // Data handling
229 u8 TwoWire_write(TwoWire *self, u8 data);
230 u8 TwoWire_writeBytes(TwoWire *self, const u8 *data, u16
    length);
231 //u8 TwoWire_write(TwoWire *self, u8 data);
232 //u8 TwoWire_writeBytes(TwoWire *self, const u8 *data, u16
    length);
233 //u8 TwoWire_writeUInt8(TwoWire *self, u8 n);
234 //u8 TwoWire_writeUInt32(TwoWire *self, u32 n);
235 //u8 TwoWire_writeInt(TwoWire *self, int n);
236 int TwoWire_available(TwoWire *self);
237 int TwoWire_read(TwoWire *self);
238 int TwoWire_peek(TwoWire *self);
239
240 void TwoWire_onReceive(TwoWire *self, void (*)(u8));
241 void TwoWire_onRequest(TwoWire *self, void (*)(void));
242
243 // Common
244 void TwoWire_end(TwoWire *self);
245 void TwoWire_flush(TwoWire *self);
246 //u8 TwoWire_specialConfig(TwoWire *self, u8 smbus1v1, u8
    longsetup, u8 sda_hold, u8 smbus1v1_dual, u8

```

```

        sda_hold_dual);
247 //void TwoWire_enableDualMode(TwoWire *self, u8 fmp_enable);
248
249 #endif /* CWIRE_H */

```

1.6.2 CWire.c

```

1  #include <stdlib.h>
2  // #include <string.h>
3  // #include <inttypes.h>
4  #include <avr/interrupt.h>
5
6  #ifndef F_CPU
7  #define F_CPU 3333333UL
8  #endif
9
10 #include "CWire.h"
11
12 #ifndef TWI0
13 #define TWI0
14 #endif
15
16 #include "twi_pins.h"
17
18 volatile u8 sleepStack = 0;
19
20 void pauseDeepSleep(u8 moduelAddr);
21 void restoreSleep(u8 moduelAddr);
22
23 TwoWire* twi0_wire;
24
25 #define TWI_BAUD(freq, t_rise) ((F_CPU / freq) / 2) - (5 +
    (((F_CPU / 1000000) * t_rise) / 2000))
26 u8 MasterCalcBaud(u32 freq) {
27     i16 baud;
28     if (freq >= 600000) { // assuming 1.5kOhm
29         baud = TWI_BAUD(freq, 250);
30     } else if (freq >= 400000) { // assuming 2.2kOhm
31         baud = TWI_BAUD(freq, 400);
32     } else { // assuming 4.7kOhm
33         baud = TWI_BAUD(freq, 600);
34     }
35
36     const u8 baudlimit = 0;
37
38     if (baud < baudlimit) {
39         return baudlimit;

```



```

40         } else if (baud > 255) {
41             return 255;
42         }
43
44         return (u8)baud;
45     }
46
47     void TwoWire_init(TwoWire *self, TWI_t *twi_module) {
48         self->_module = twi_module;
49         twi0_wire = self;
50     }
51
52     u8 TwoWire_pins(TwoWire *self, u8 sda, u8 scl) {
53         return TWI0_Pins(sda, scl);
54     }
55
56     void TwoWire_usePullups(TwoWire *self) {
57         TWI0_usePullups();
58     }
59
60     u8 TwoWire_setClock(TwoWire *self, u32 freq) {
61         TWI_t* module = self->_module;
62         if (__builtin_constant_p(freq)) {
63             if ((freq < 1000) || (freq > 15000000)) {
64                 return 1;
65             }
66         } else {
67             if (freq < 1000) {
68                 return 1;
69             }
70         }
71         if (self->_bools._hostEnabled == 1) {
72             u8 newBaud = MasterCalcBaud(freq);
73             u8 oldBaud = module->MCTRLA;
74             if (newBaud != oldBaud) {
75                 u8 restore = module->MCTRLA;
76                 module->MCTRLA = 0;
77                 module->MBAUD = newBaud;
78                 if (freq > 400000) {
79                     module->CTRLA |= TWI_FMPEN_bm;
80                 } else {
81                     module->CTRLA &= ~TWI_FMPEN_bm;
82                 }
83                 module->MCTRLA = restore;
84                 if (restore & TWI_ENABLE_bm) {
85                     module->MSTATUS = TWI_BUSSTATE_IDLE_gc;
86                 }
87             }
88             return 0;

```

```

89         }
90         return 1;
91     }
92
93     //Master
94     void TwoWire_Master_begin(TwoWire *self) {
95         TWI0_ClearPins();
96
97         self->_bools._hostEnabled = 1;
98         TWI_t* module = self->_module;
99         module->MCTRLA = TWI_ENABLE_bm;
100        module->MSTATUS = TWI_BUSSTATE_IDLE_gc;
101
102        TwoWire_setClock(self, DEFAULT_FREQUENCY);
103    }
104
105    void TwoWire_endMaster(TwoWire *self) {
106        if (true == self->_bools._hostEnabled) {
107            self->_module->MCTRLA = 0x00;
108            self->_module->MBAUD = 0x00;
109            self->_bools._hostEnabled = 0x00;
110        }
111    }
112
113    void TwoWire_beginTransmission(TwoWire *self, u8 addr) {
114        if (__builtin_constant_p(addr) > 0x7F) {
115            badArg("Supplied address seems to be 8 bit. Only 7 bit
116                addresses are supported");
117            return;
118        }
119        if (self->_bools._hostEnabled) {
120            self->_clientAddress = addr << 1;
121            self->_bytesToReadWrite = 0;
122            self->_bytesReadWritten = 0;
123        }
124    }
125    u8 TwoWire_endTransmission(TwoWire *self, u8 sendStop) {
126        return TwoWire_masterTransmit(self,
127            &self->_bytesToReadWrite, self->_hostBuffer,
128            self->_clientAddress, sendStop);
129    }
130
131    twi_buf_index_t TwoWire_requestFrom(TwoWire *self, u8 addr,
132        twi_buf_index_t qty, u8 sendStop) {
133        if (__builtin_constant_p(qty)) {
134            if (qty > TWI_BUFFER_LENGTH) {
135                badArg("requestFrom requests more bytes than buffer
136                    space");
137            }
138        }
139    }

```

```

133     }
134     if (qty >= TWI_BUFFER_LENGTH) {
135         qty = TWI_BUFFER_LENGTH;
136     }
137
138     self->_clientAddress = addr << 1;
139     self->_bytesToReadWrite = qty;
140     self->_bytesReadWritten = 0;
141
142     return TwoWire_masterReceive(self,
        &self->_bytesToReadWrite, self->_hostBuffer,
        self->_clientAddress, sendStop);
143 }
144
145 u8 TwoWire_masterTransmit(TwoWire *self, twi_buf_index_t
    *length, u8 *buff, u8 addr, u8 sendStop) {
146     TWI_t* module = self->_module;
147     __asm__ __volatile__("\n\t" : "+z"(module));
148
149     TWI_INIT_ERROR;
150     u8 currentSM;
151     u8 currentStatus;
152     u8 stat = 0;
153
154     u16 dataToWrite = *length;
155     #if defined (TWI_TIMEOUT_ENABLE)
156     u16 timeout = (F_CPU/1000);
157     #endif
158
159     if ((module->MCTRLA & TWI_ENABLE_bm) == 0x00) {
160         return TWI_ERR_UNINIT;
161     }
162
163     while (1) {
164         currentStatus = module->MSTATUS;
165         currentSM = currentStatus & TWI_BUSSTATE_gm;
166
167         if (currentSM == TWI_BUSSTATE_UNKNOWN_gc) {
168             return TWI_ERR_UNINIT;
169         }
170
171         #if defined (TWI_TIMEOUT_ENABLE)
172         if (--timeout == 0) {
173             if (currentSM == TWI_BUSSTATE_OWNER_gc) {
174                 TWI_SET_ERROR(TWI_ERR_TIMEOUT);
175             } else if (currentSM == TWI_BUSSTATE_IDLE_gc) {
176                 TWI_SET_ERROR(TWI_ERR_PULLUP);
177             } else {
178                 TWI_SET_ERROR(TWI_ERR_UNDEFINED);

```

```

179         }
180         break;
181     }
182     #endif
183
184     if (currentStatus & TWI_ARBLOST_bm) {
185         TWI_SET_ERROR(TWI_ERR_BUS_ARB);
186         break;
187     }
188
189     if (currentSM != TWI_BUSSTATE_BUSY_gc) {
190         if (stat == 0x00) {
191             module->MADDR = ADD_WRITE_BIT(addr);
192             stat |= 0x01;
193             #if defined (TWI_TIMEOUT_ENABLE)
194                 timeout = (F_CPU/1000);
195             #endif
196         } else {
197             if (currentStatus & TWI_WIF_bm) {
198                 if (currentStatus & TWI_RXACK_bm) {
199                     if (stat & 0x02) {
200                         if (dataToWrite != 0) {
201                             TWI_SET_ERROR(TWI_ERR_ACK_DAT);
202                         }
203                     } else {
204                         TWI_SET_ERROR(TWI_ERR_ACK_ADR);
205                     }
206                     break;
207                 } else {
208                     if (dataToWrite != 0) {
209                         module->MDATA = *buff;
210                         buff++;
211                         dataToWrite--;
212                         stat |= 0x02;
213                         #if defined (TWI_TIMEOUT_ENABLE)
214                             timeout = (F_CPU/1000);
215                         #endif
216                     } else {
217                         break;
218                     }
219                 }
220             }
221         }
222     }
223 }
224
225 *length -= dataToWrite;
226 if ((sendStop != 0) || (TWI_ERR_SUCCESS != TWI_GET_ERROR)) {
227     module->MCTRLB = TWI_MCMD_STOP_gc;

```

```

228     }
229     return TWI_GET_ERROR;
230 }
231
232 u8 TwoWire_masterReceive(TwoWire *self, u16 *length, u8 *buff,
    u8 addr, u8 sendStop) {
233     TWI_t *module = self->_module;
234     __asm__ __volatile__("\n\t" : "+z"(module));
235
236     TWIR_INIT_ERROR;
237     u16 dataToRead = *length;
238
239     u8 currentSM;
240     u8 currentStatus;
241     u8 state = 0;
242     #if defined (TWI_TIMEOUT_ENABLE)
243     u16 timeout = (F_CPU/1000);
244     #endif
245
246     while(1) {
247         currentStatus = module->MSTATUS;
248         currentSM = currentStatus & TWI_BUSSTATE_gm;
249
250         if (currentSM == TWI_BUSSTATE_UNKNOWN_gc) {
251             TWIR_SET_ERROR(TWI_ERR_UNINIT);
252             break;
253         }
254
255         #if defined (TWI_TIMEOUT_ENABLE)
256         if (--timeout == 0) {
257             if (currentSM == TWI_BUSSTATE_OWNER_gc) {
258                 TWIR_SET_ERROR(TWI_ERR_TIMEOUT);
259             } else if (currentSM == TWI_BUSSTATE_IDLE_gc) {
260                 TWIR_SET_ERROR(TWI_ERR_PULLUP);
261             } else {
262                 TWIR_SET_ERROR(TWI_ERR_UNDEFINED);
263             }
264             break;
265         }
266         #endif
267
268         if (currentStatus & TWI_ARBLOST_bm) {
269             TWIR_SET_ERROR(TWI_ERR_BUS_ARB);
270             break;
271         }
272
273         if (currentSM != TWI_BUSSTATE_BUSY_gc) {
274             if (state == 0x00) {
275                 module->MADDR = ADD_READ_BIT(addr);

```

```

276         state |= 0x01;
277         #if defined (TWI_TIMEOUT_ENABLE)
278         timeout = (F_CPU / 1000);
279         #endif
280     } else {
281         if (currentStatus & TWI_WIF_bm) {
282             TWIR_SET_ERROR(TWI_ERR_ACK_ADR);
283             module->MCTRLB = TWI_MCMD_STOP_gc;
284             break;
285         } else if (currentStatus & TWI_RIF_bm) {
286             *buff = module->MDATA;
287             buff++;
288             #if defined (TWI_TIMEOUT_ENABLE)
289             timeout = (F_CPU / 1000);
290             #endif
291             if (--dataToRead != 0) {
292                 module->MCTRLB = TWI_MCMD_RECVTRANS_gc;
293             } else {
294                 if (sendStop != 0) {
295                     module->MCTRLB = TWI_ACKACT_bm |
296                         TWI_MCMD_STOP_gc;
297                 } else {
298                     module->MCTRLB = TWI_ACKACT_bm;
299                 }
300                 break;
301             }
302         }
303     }
304 }
305 *length -= dataToRead;
306 return TWIR_GET_ERROR;
307 }
308
309 // Slave
310 void TwoWire_Slave_begin(TwoWire *self, u8 addr, u8 bc, u8
311     sec_addr) {
312     if (__builtin_constant_p(addr)) {
313         if (addr > 0x7F) {
314             badArg("TWI addresses must be supplied in 7-bit
315                 format");
316             return;
317         }
318     }
319     #if defined(TWI_MANDS)
320     if (self->_bools._clientEnabled == 1) {
321         return;
322     }

```

```

322         #else
323         if ((self->_bools._hostEnabled |
324             self->_bools._clientEnabled) == 1) {
325             return;
326         }
327         #endif
328
329         TWI0_ClearPins();
330
331         self->_bools._clientEnabled = 1;
332         self->client_irq_mask= TWI_COLL_bm;
333         TWI_t* module = self->_module;
334         module->SADDR = (addr << 1) | bc; //broadcast
335         module->SADDRMASK = sec_addr;
336         module->SCTRLA = TWI_DIEN_bm | TWI_APIEN_bm | TWI_PIEN_bm |
337             TWI_ENABLE_bm;
338     }
339
340     void TwoWire_endSlave(TwoWire *self) {
341         if (self->_bools._clientEnabled == 1) {
342             self->_module->SADDR = 0x00;
343             self->_module->SCTRLA = 0x00;
344             self->_module->SADDRMASK = 0x00;
345             self->_bools._clientEnabled = 0x00;
346         }
347     }
348
349     u8 TwoWire_getIncomingAddress(TwoWire *self) {
350         #if defined (TWI_MANDS)
351         return self->_incomingAddress;
352         #else
353         return self->_clientAddress;
354         #endif
355     }
356
357     twi_buf_index_t TwoWire_getBytesRead(TwoWire *self) {
358         twi_buf_index_t num = self->_bytesTransmittedS;
359         self->_bytesTransmittedS = 0;
360         return num;
361     }
362
363     u8 TwoWire_slaveTransactionOpen(TwoWire *self) {
364         u8 status = self->_module->SSTATUS;
365         if (!(status & TWI_AP_bm)) {
366             return 0;
367         }
368         if (status & TWI_DIR_bm) {
369             return 2;
370         }

```

```

369         return 1;
370     }
371
372     u8 TwoWire_checkPinLevels(TwoWire *self) {
373         return TWIO_checkPinLevel();
374     }
375
376     void TwoWire_onReceive(TwoWire *self, void (*function)(u8)) {
377         self->user_onReceive = function;
378     }
379
380     void TwoWire_onRequest(TwoWire *self, void (*function)(void)) {
381         self->user_onRequest = function;
382     }
383
384     ISR(TWIO_TWIS_vect) {
385         TwoWire_HandleSlaveIRQ(twi0_wire);
386     }
387
388     void TwoWire_HandleSlaveIRQ(TwoWire *wire_s) {
389         if (wire_s == NULL) {
390             return;
391         }
392
393         u8 *address, *buffer;
394         twi_buf_index_t *head, *tail;
395         #if defined(TWI_MANDS)
396         address = &(wire_s->_incomingAddress);
397         head    = &(wire_s->_bytesToReadWriteS);
398         tail    = &(wire_s->_bytesReadWrittenS);
399         buffer = wire_s->_clientBuffer;
400         #else
401         address = &(wire_s->_clientAddress);
402         head    = &(wire_s->_bytesToReadWrite);
403         tail    = &(wire_s->_bytesReadWritten);
404         buffer = wire_s->_hostBuffer;
405         #endif
406
407         #if defined(TWI_MANDS)
408         wire_s->_bools._toggleStreamFn = 0x01;
409         #endif
410
411         u8 action = 0;
412         u8 clientStatus = wire_s->_module->SSTATUS;
413
414
415         if (clientStatus & TWI_APIF_bm) { // Address/Stop Bit set
416             if (wire_s->_bools._hostDataSent != 0) { // At this
417                 point, we have either a START, REPSTART or a STOP

```



```

417         wire_s->_bools._hostDataSent = 0x00;
418         if (wire_s->user_onReceive != NULL) { // only if the
419             last APIF was a Master Write,
420             wire_s->user_onReceive((*head)); // we notify the
421             sketch about new Data
422         }
423     }
424     if (clientStatus & TWI_AP_bm) { // Address bit set
425         if ((*head) == 0) { // only if there was
426             no data (START)
427             pauseDeepSleep((u8)((u16)wire_s->_module)); // Only
428             START can wake from deep sleep, change to IDLE
429         }
430         (*head) = 0;
431         (*tail) = 0;
432         (*address) = wire_s->_module->SDATA; // read address
433         from data register
434         if (clientStatus & TWI_DIR_bm) { // Master is reading
435             if (wire_s->user_onRequest != NULL) {
436                 wire_s->user_onRequest();
437             }
438             if ((*head) == 0) { // If no data
439                 to transmit, send NACK
440                 action = TWI_ACKACT_bm | TWI_SCMD_COMPTRANS_gc;
441                 // NACK + "Wait for any Start (S/Sr)
442                 condition"
443             } else {
444                 action = TWI_SCMD_RESPONSE_gc; // "Execute
445                 Acknowledge Action succeeded by reception of
446                 next byte"
447             }
448         } else { // Master is writing
449             wire_s->_bools._hostDataSent = 0x01;
450             action = TWI_SCMD_RESPONSE_gc; // "Execute
451             Acknowledge Action succeeded by slave data
452             interrupt"
453         }
454     } else { // Stop bit set
455         restoreSleep((u8)((u16)wire_s->_module));
456         (*head) = 0;
457         (*tail) = 0;
458         action = TWI_SCMD_COMPTRANS_gc; // "Wait for any Start
459         (S/Sr) condition"
460     }
461 } else if (clientStatus & TWI_DIF_bm) { // Data bit set
462     if (clientStatus & TWI_DIR_bm) { // Master is reading
463         if (clientStatus & wire_s->client_irq_mask) { // If a
464             collision was detected, or RXACK bit is set (when

```

```

        it matters)
452 wire_s->client_irq_mask = TWI_COLL_bm; // stop
        checking for NACK
453 (*head) = 0; // Abort
        further data writes
454 action = TWI_SCMD_COMPTRANS_gc; // "Wait for
        any Start (S/Sr) condition"
455 } else { // RXACK bit not
        set, no COLL
456 wire_s->bytesTransmittedS++; // increment
        bytes transmitted counter (for register model)
457 wire_s->client_irq_mask = TWI_COLL_bm |
        TWI_RXACK_bm; // start checking for NACK
458 if ((*tail) < (*head)) { // Data is
        available
459 wire_s->module->SDATA = buffer[(*tail)]; //
        Writing to the register to send data
460 (*tail)++; // Increment
        counter for sent bytes
461 action = TWI_SCMD_RESPONSE_gc; // "Execute
        Acknowledge Action succeeded by reception of
        next byte"
462 } else { // No more
        data available
463 (*head) = 0; // Avoid
        triggering REPSTART handler
464 action = TWI_SCMD_COMPTRANS_gc; // "Wait for
        any Start (S/Sr) condition"
465 }
466 }
467 } else { // Master is
        writing
468 uint8_t payload = wire_s->module->SDATA; // reading
        SDATA will clear the DATA IRQ flag
469 if ((*head) < TWI_BUFFER_LENGTH) { // make sure
        that we don't have a buffer overflow in case
        Master ignores NACK
470 buffer[(*head)] = payload; // save data
471 (*head)++; // Advance
        Head
472 if ((*head) == TWI_BUFFER_LENGTH) { // if
        buffer is not yet full
473 action = TWI_ACKACT_bm | TWI_SCMD_COMPTRANS_gc;
        // "Execute ACK Action succeeded by waiting
        for any Start (S/Sr) condition"
474 } else { // else
        buffer would overflow with next byte
475 action = TWI_SCMD_RESPONSE_gc; //
        "Execute Acknowledge Action succeeded by

```

```

476         }
477     }
478 }
479 }
480 wire_s->_module->SCTRLB = action; // using local variable
      (register) reduces the amount of loading _module
481 #if defined(TWI_MANDS)
482 wire_s->_bools._toggleStreamFn = 0x00;
483 #endif
484 }
485
486 // Data handling
487 u8 TwoWire_write(TwoWire *self, u8 data) {
488     u8 *txBuffer;
489     twi_buf_index_t *txHead;
490
491     #if defined(TWI_MANDS) // If host and client are split
492     if (self->_bools._toggleStreamFn == 0x01) {
493         txHead = &(self->_bytesToReadWriteS);
494         txBuffer = self->_clientBuffer;
495     } else
496     #endif
497     {
498         txHead = &(self->_bytesToReadWrite);
499         txBuffer = self->_hostBuffer;
500     }
501
502     // Put byte in txBuffer
503     if ((*txHead) < TWI_BUFFER_LENGTH) { // while buffer not
      full, write to it
504         txBuffer[(*txHead)] = data;      // Load data into the
      buffer
505         (*txHead)++;      // advancing the head
506         return 1;
507     } else {
508         return 0; // Buffer full
509     }
510 }
511
512 // Function to write an array of bytes
513 u8 TwoWire_writeBytes(TwoWire *self, const u8 *data, u16
      length) {
514     u16 i = 0;
515     for (; i < length; i++) {
516         if (TwoWire_write(self, data[i]) == 0) {
517             break; // Stop if buffer is full
518         }
519     }

```

```

520         return i; // Number of bytes successfully written
521     }
522
523
524     int TwoWire_available(TwoWire *self) {
525         return self->_bytesToReadWrite - self->_bytesReadWritten;
526     }
527
528     int TwoWire_read(TwoWire *self) {
529         uint8_t *rxBuffer;
530         twi_buf_index_t *rxHead, *rxTail;
531         #if defined(TWI_MANDS) // Add following
532             if host and client are split
533             if (_bools._toggleStreamFn == 0x01) {
534                 rxHead = &(_bytesToReadWriteS);
535                 rxTail = &(_bytesReadWrittenS);
536                 rxBuffer = _clientBuffer;
537             } else
538             #endif
539             {
540                 rxHead = &(self->_bytesToReadWrite);
541                 rxTail = &(self->_bytesReadWritten);
542                 rxBuffer = self->_hostBuffer;
543             }
544
545             if ((*rxTail) < (*rxHead)) { // if there are bytes to read
546                 uint8_t c = rxBuffer[(*rxTail)];
547                 (*rxTail)++;
548                 return c;
549             } else { // No bytes to read. At this
550                 point, rxTail moved up to
551                 return -1; // rxHead. To reset both to 0,
552                 a MasterRead or AddrWrite has to be called
553             }
554         }
555
556         int TwoWire_peek(TwoWire *self) {
557             uint8_t *rxBuffer;
558             twi_buf_index_t *rxHead, *rxTail;
559             #if defined(TWI_MANDS) // Add following
560                 if host and client are split
561                 if (_bools._toggleStreamFn == 0x01) {
562                     rxHead = &(_bytesToReadWriteS);
563                     rxTail = &(_bytesReadWrittenS);
564                     rxBuffer = _clientBuffer;
565                 } else
566                 #endif
567                 {

```

```

565         rxHead = &(self->_bytesToReadWrite);
566         rxTail = &(self->_bytesReadWritten);
567         rxBuffer = self->_hostBuffer;
568     }
569
570     if ((*rxTail) < (*rxHead)) { // if there are bytes to read
571         return rxBuffer[(*rxTail)];
572     } else { // No bytes to read
573         return -1;
574     }
575 }
576
577 // Common
578 void TwoWire_end(TwoWire *self) {
579     TwoWire_endSlave(self);
580     TwoWire_endMaster(self);
581 }
582
583 void TwoWire_flush(TwoWire *self) {
584     TWI_t* module = self->_module;
585     #if defined(ERRATA_TWI_FLUSH)
586     u8 temp_MCTRLA = module->MCTRLA;
587     u8 temp_SCTRLA = module->SCTRLA;
588     module->MCTRLA = 0x00;
589     module->SCTRLA = 0x00;
590     module->MCTRLA = temp_MCTRLA;
591     module->MSTATUS = 0x01;
592     module->SCTRLA = temp_SCTRLA;
593     #else
594     module->MCTRLB |= TWI_FLUSH_bm;
595     #endif
596 }
597
598 void pauseDeepSleep(uint8_t module_lower_Addr) {
599     #if defined(TWI_USING_WIRE1)
600     uint8_t bit_mask = 0x10;
601     if (module_lower_Addr == (uint8_t)((uint16_t)&TWI1)){
602         bit_mask = 0x20;
603     }
604     uint8_t sleepStackLoc = sleepStack;
605     if (sleepStackLoc == 0) { // Save sleep state only if
606         stack empty
607         sleepStackLoc = SLPCTRL.CTRLA; // save sleep settings
608         to sleepStack
609         SLPCTRL.CTRLA = sleepStackLoc & 0x01; // only leave the
610         SEN bit, if it was set
611     }
612     sleepStackLoc |= bit_mask; // Remember which module is
613     busy

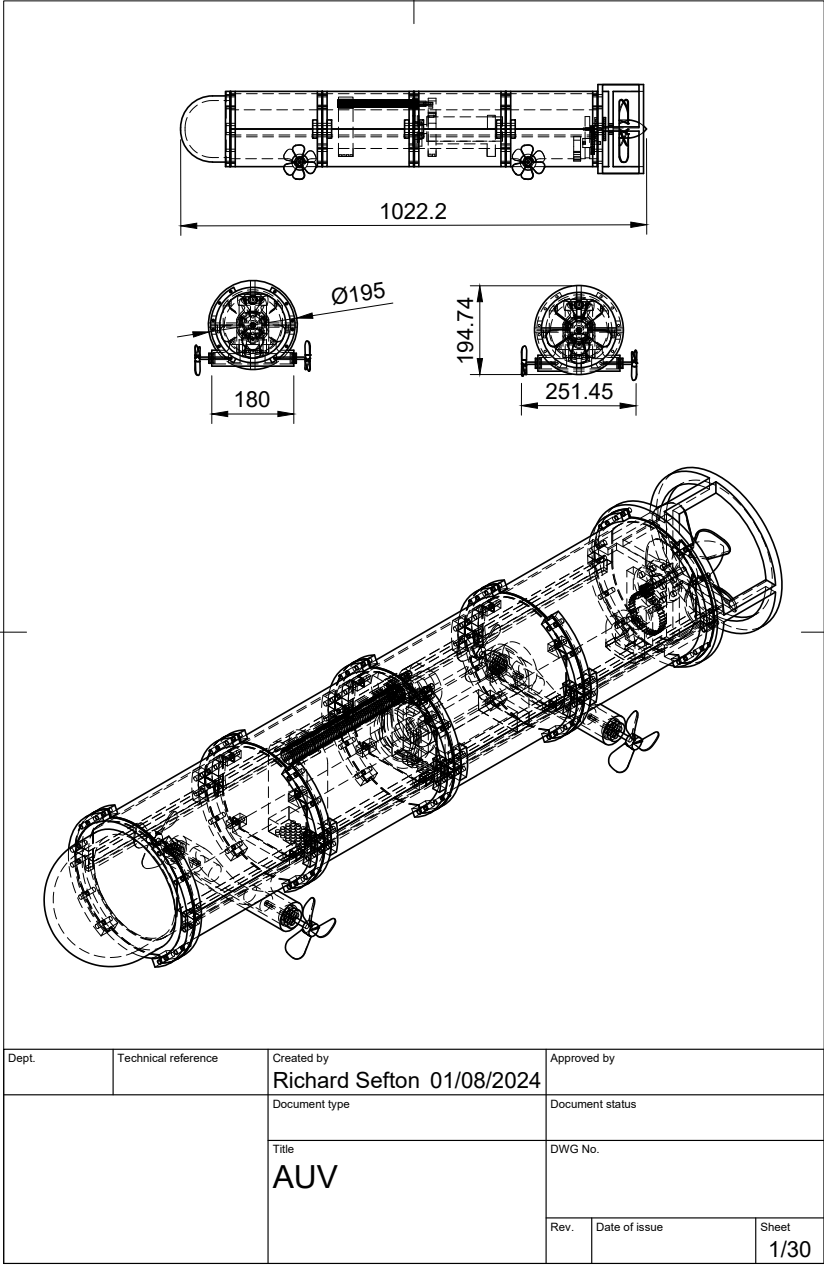
```

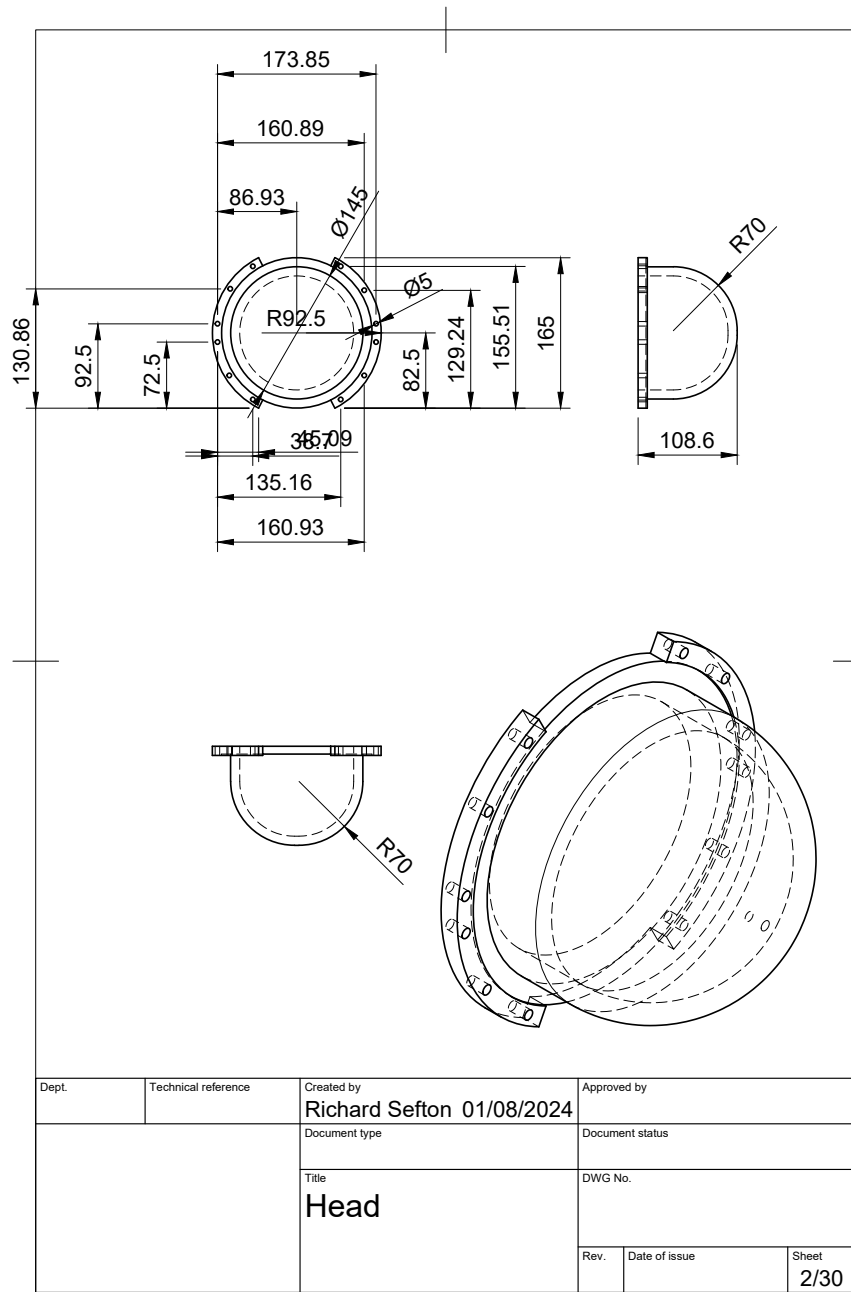
```

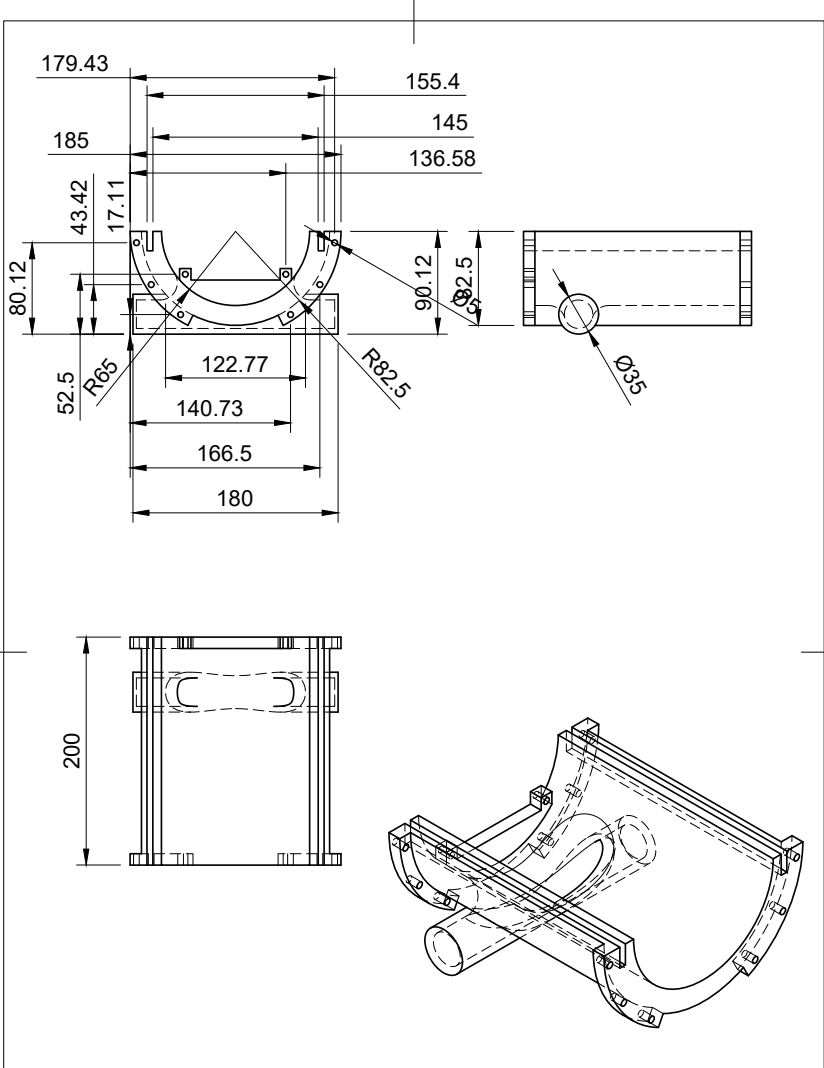
610         sleepStack = sleepStackLoc;
611     #else
612         (void) module_lower_Addr;
613
614         if (sleepStack == 0x00) {
615             uint8_t slp = SLPCTRL.CTRLA; // save current sleep State
616             sleepStack = slp;             // using local variable for
                                           less store/load
617             SLPCTRL.CTRLA = slp & 0x01; // only leave the SEN bit,
                                           if it was set
618         }
619     #endif
620 }
621
622 void restoreSleep(uint8_t module_lower_Addr) {
623     #if defined(TWI_USING_WIRE1)
624         uint8_t bit_mask = ~0x10;
625         if (module_lower_Addr == (uint8_t)((uint16_t)&TWI1)){
626             bit_mask = ~0x20;
627         }
628         uint8_t sleepStackLoc = sleepStack;
629         sleepStackLoc &= bit_mask;
630         if (sleepStackLoc > 0) { // only do something if sleep
                                was enabled
631             if (sleepStackLoc < 0x10) { // if upper nibble is clear
632                 SLPCTRL.CTRLA = sleepStackLoc; // restore sleep
633                 sleepStackLoc = 0;             // reset everything
634             }
635         }
636         sleepStack = sleepStackLoc;
637     #else
638         (void) module_lower_Addr;
639         SLPCTRL.CTRLA = sleepStack;
640         sleepStack = 0;
641     #endif
642 }

```

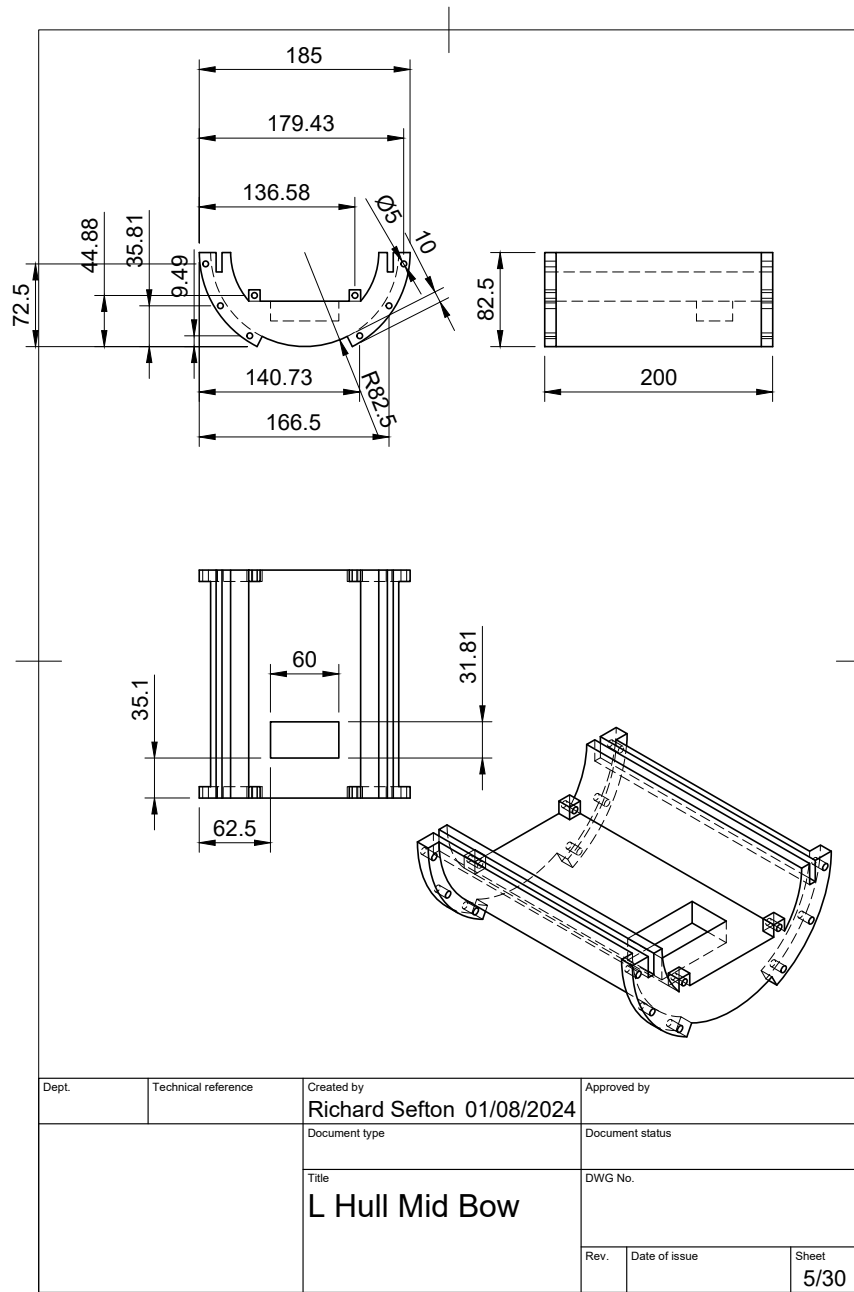
1.7 AUV Designs



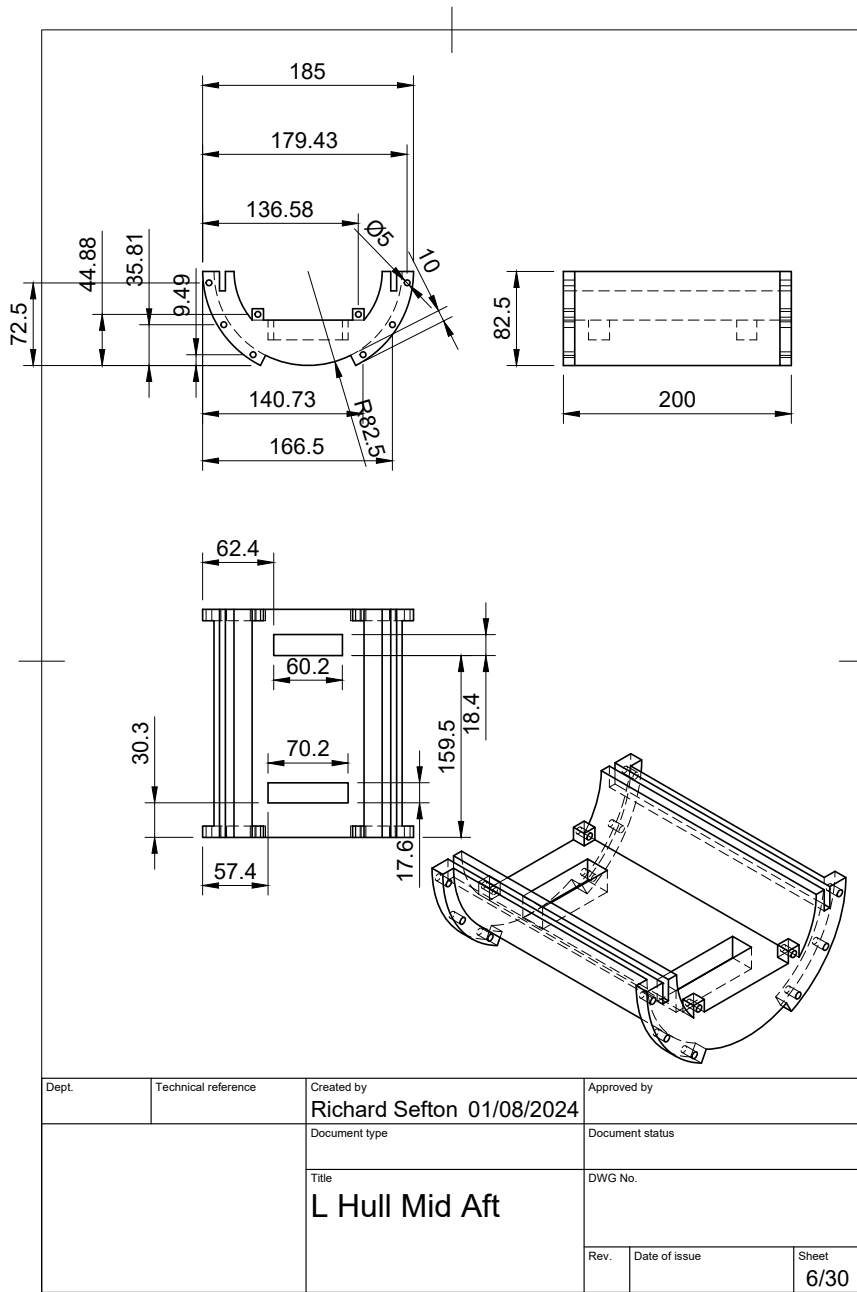


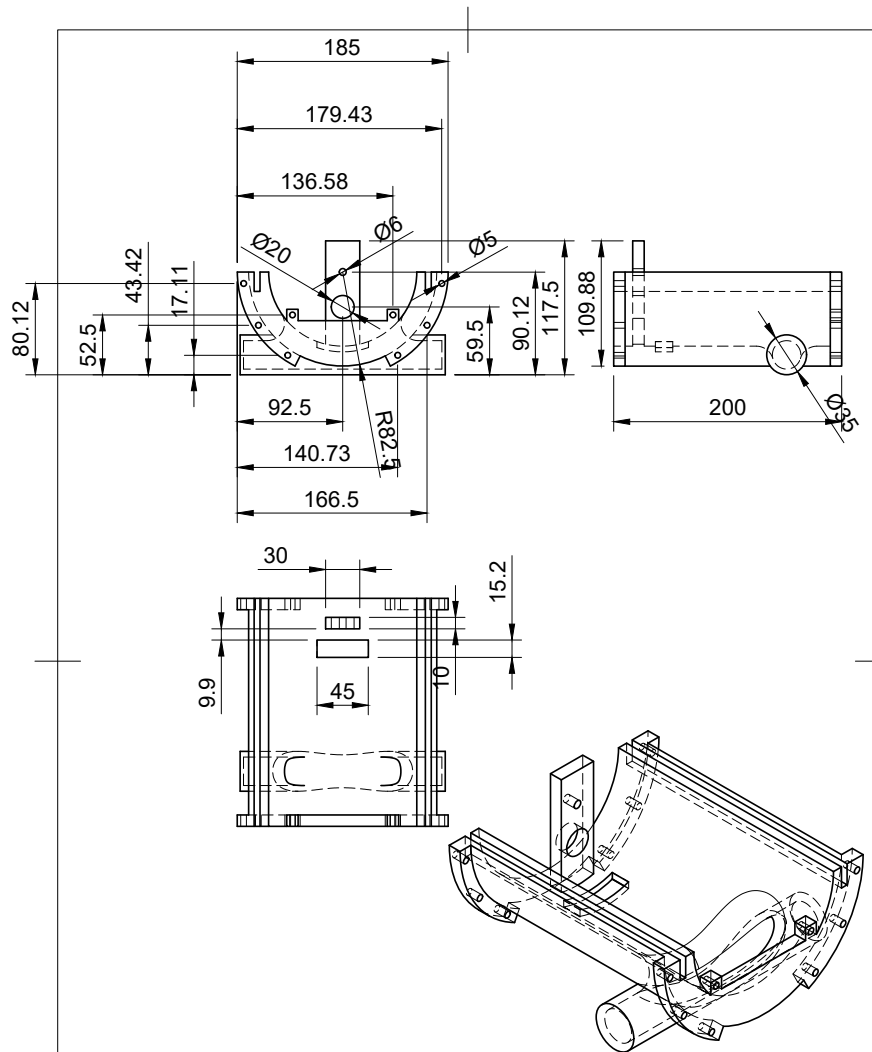


Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title L Hull Bow	DWG No.
		Rev.	Date of issue
			Sheet 4/30

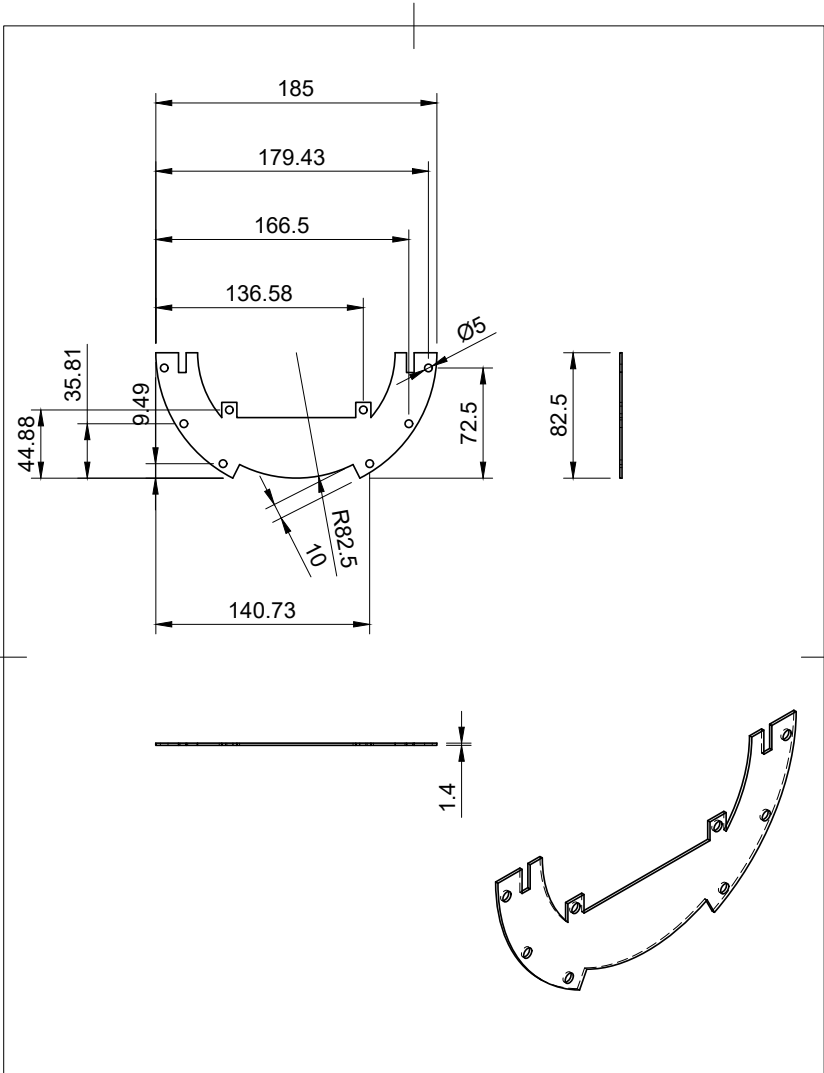


Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title L Hull Mid Bow	DWG No.
		Rev.	Date of issue
			Sheet 5/30

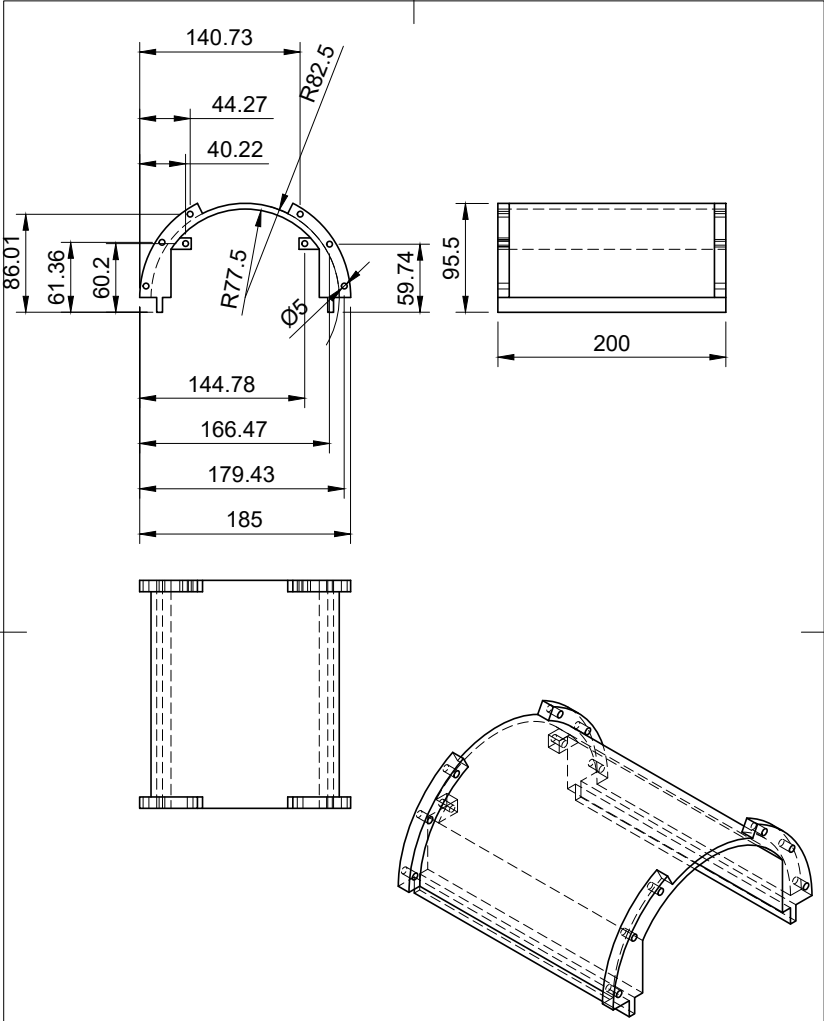




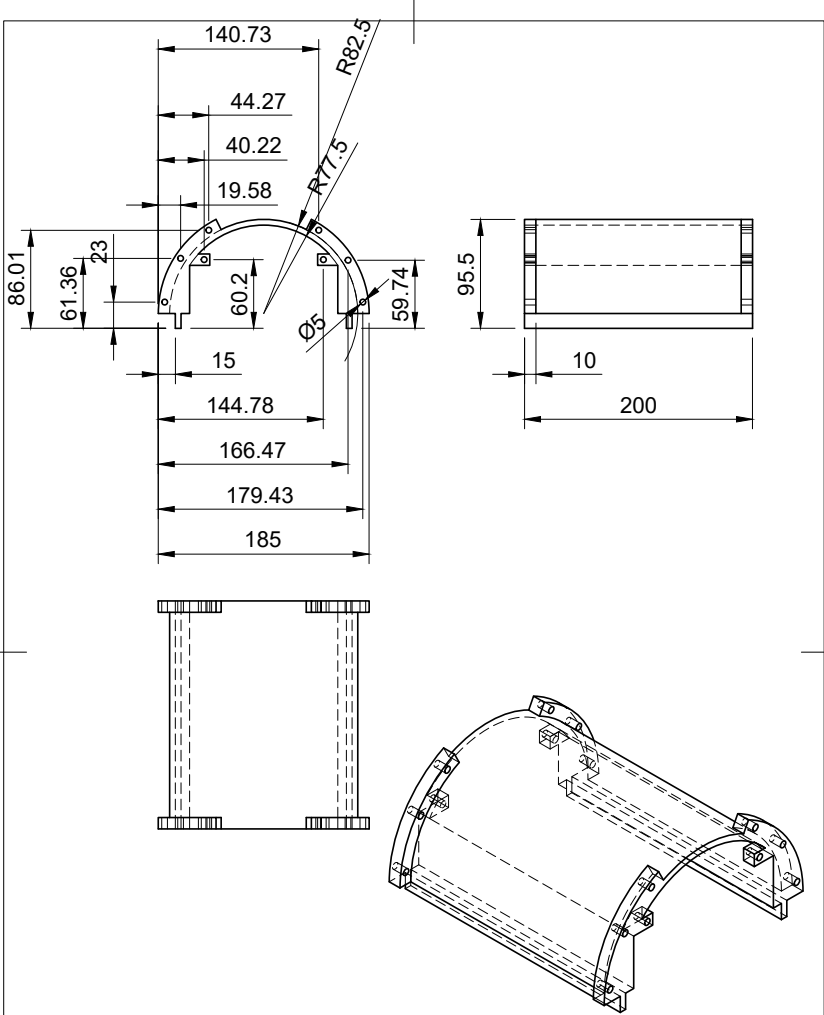
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title L Hull Mid Aft	DWG No.
		Rev.	Date of issue
			Sheet 7/30



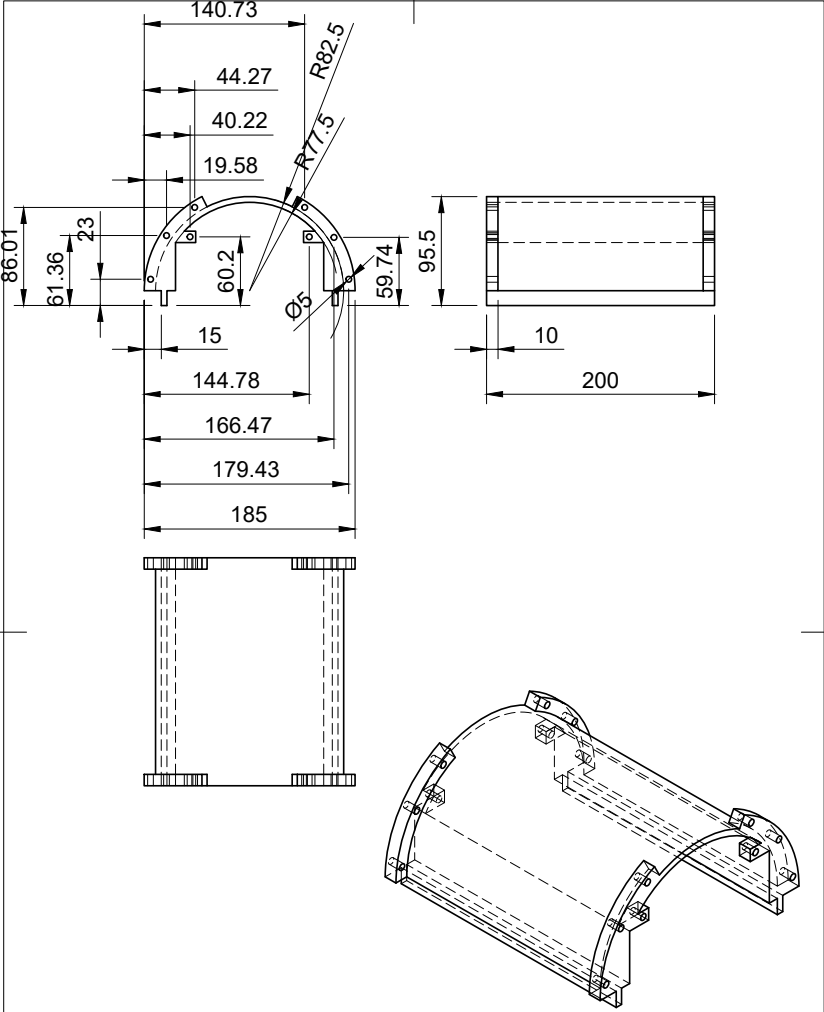
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title L Hull Gasket	DWG No.
		Rev.	Date of issue
		Sheet 8/30	



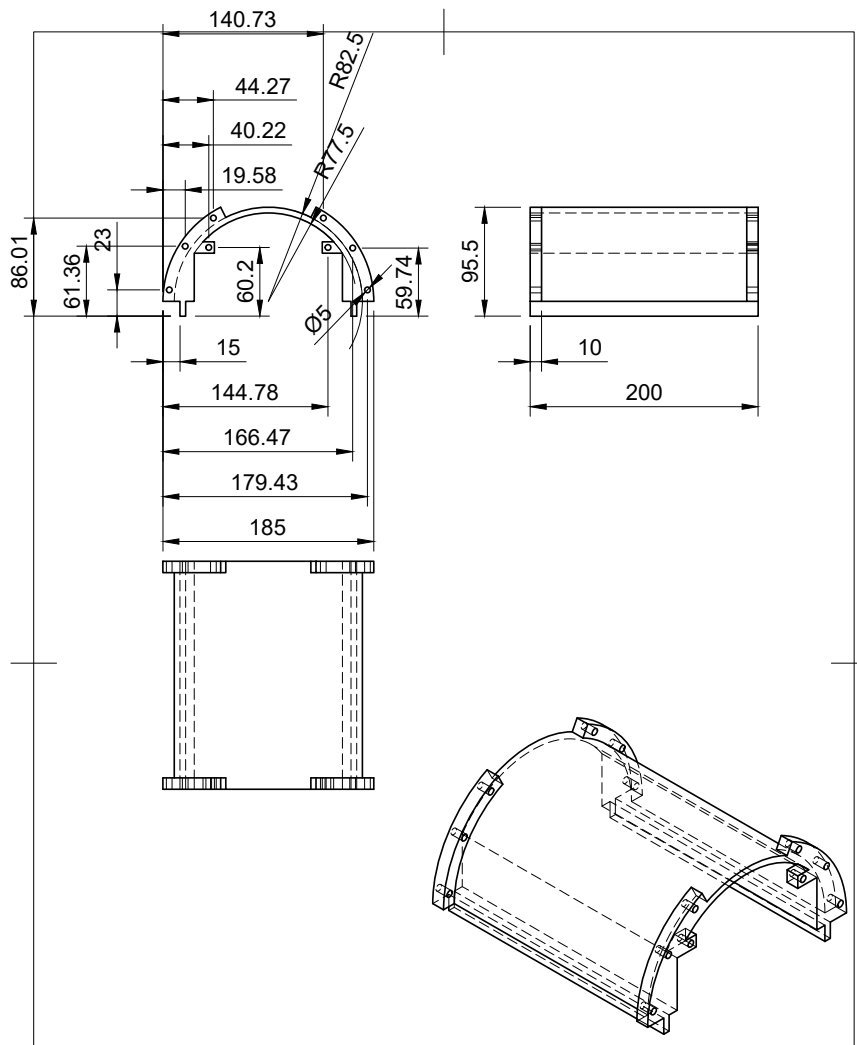
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title U Hull Bow	DWG No.
		Rev.	Date of issue
			Sheet 9/30



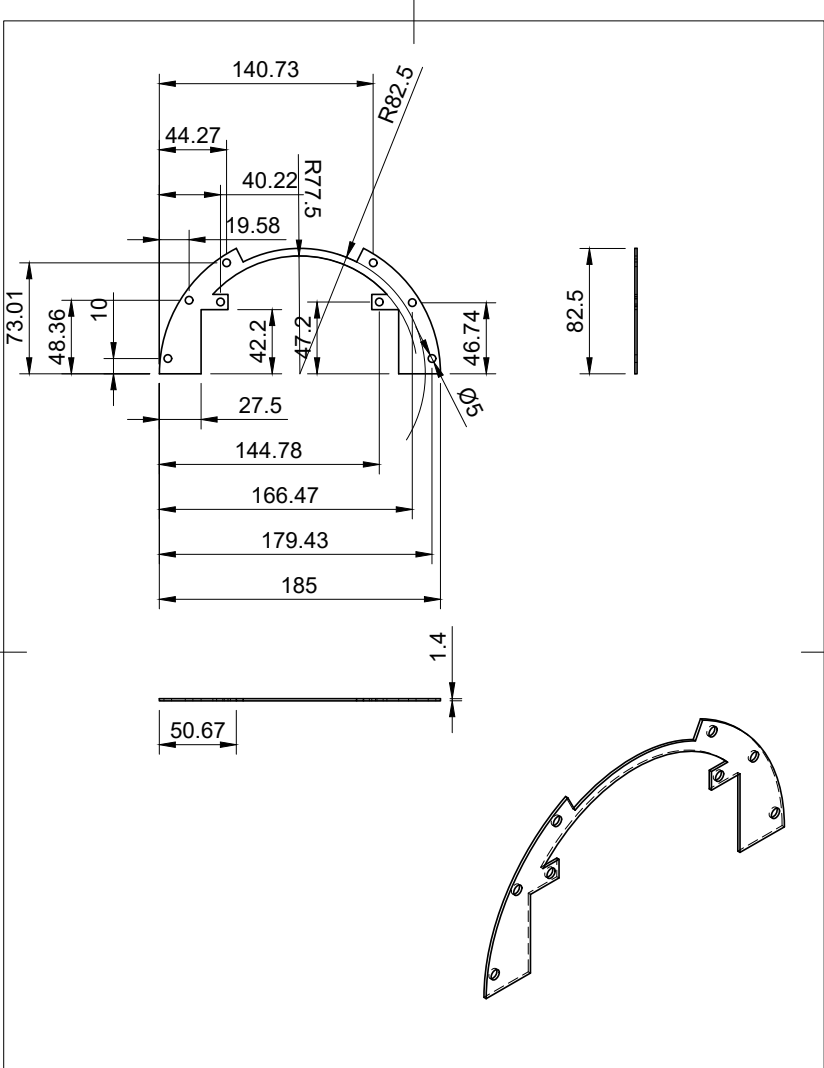
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title U Hull Mid Bow	DWG No.
		Rev.	Date of issue
			Sheet 10/30



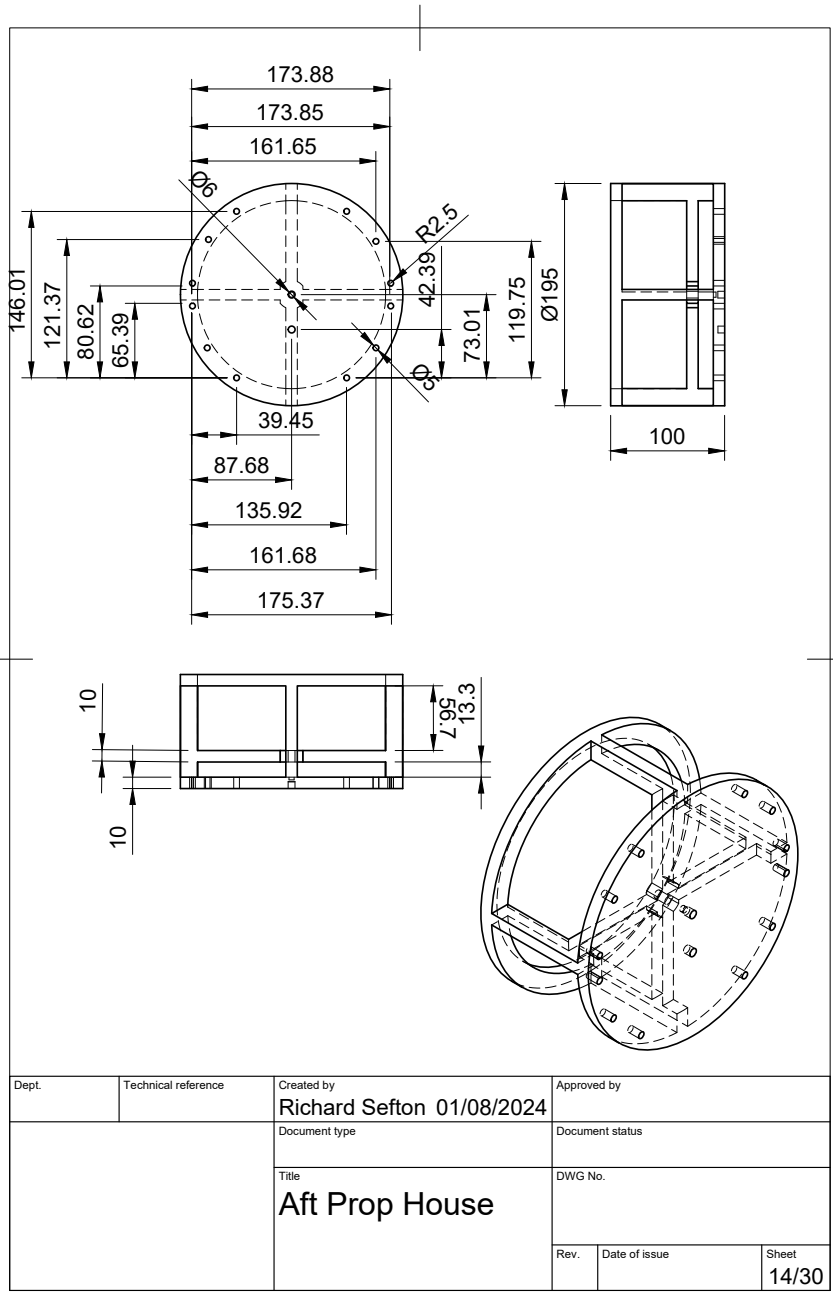
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title U Hull Mid Aft	DWG No.
		Rev.	Date of issue
		Sheet	11/30



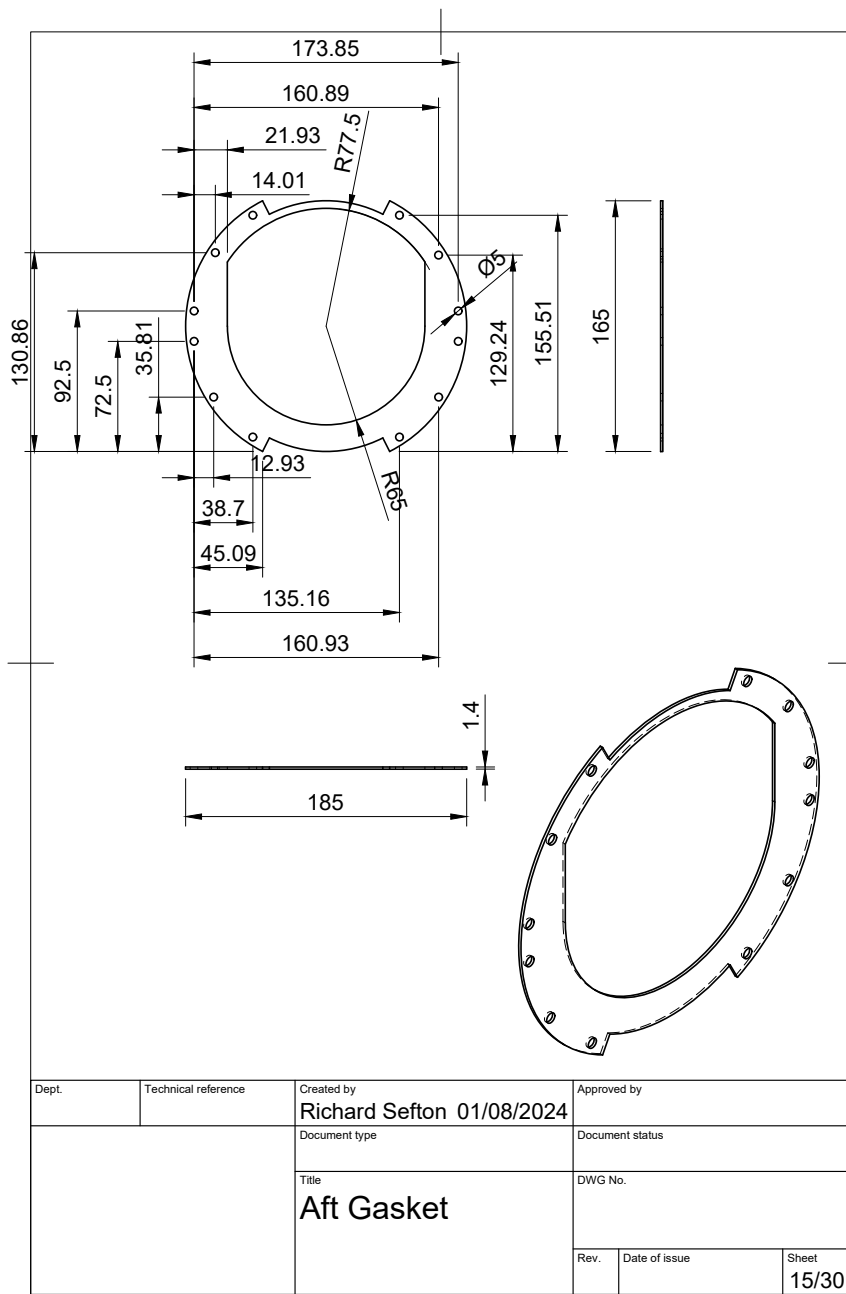
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by	
		Document type	Document status	
		Title U Hull Aft	DWG No.	
			Rev.	Date of issue
			Sheet	12/30

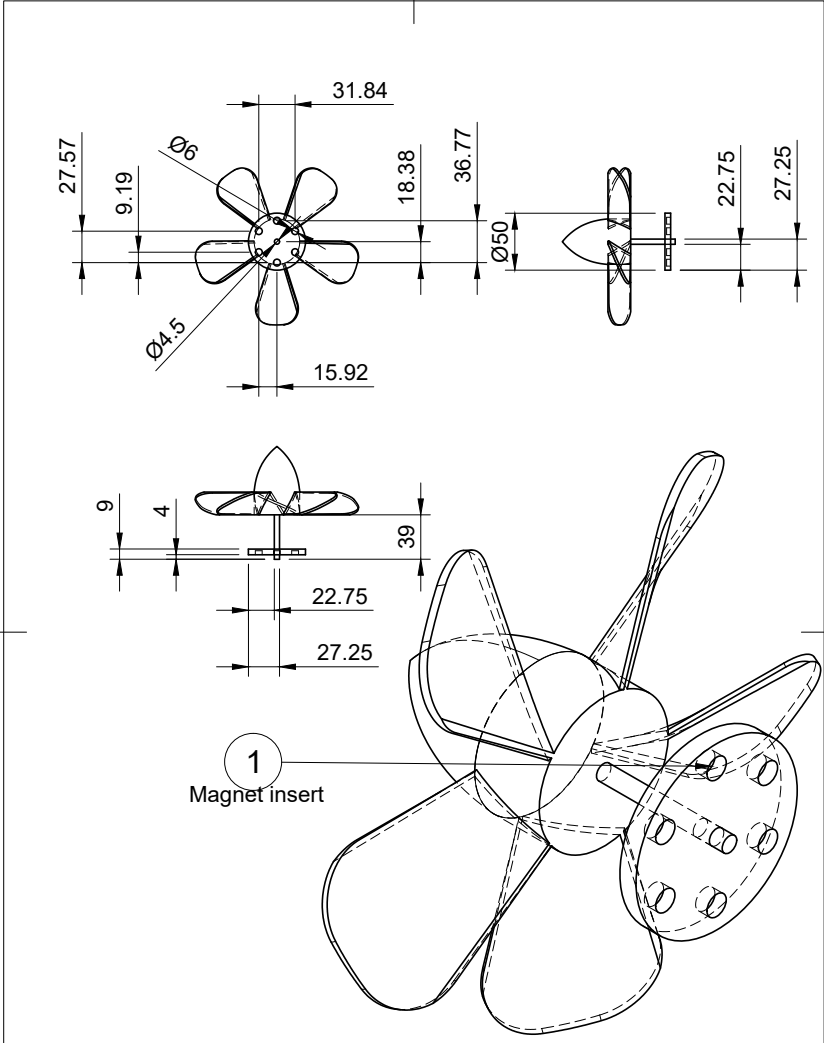


Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title U Hull Gasket	DWG No.
		Rev.	Date of issue
			Sheet 13/30

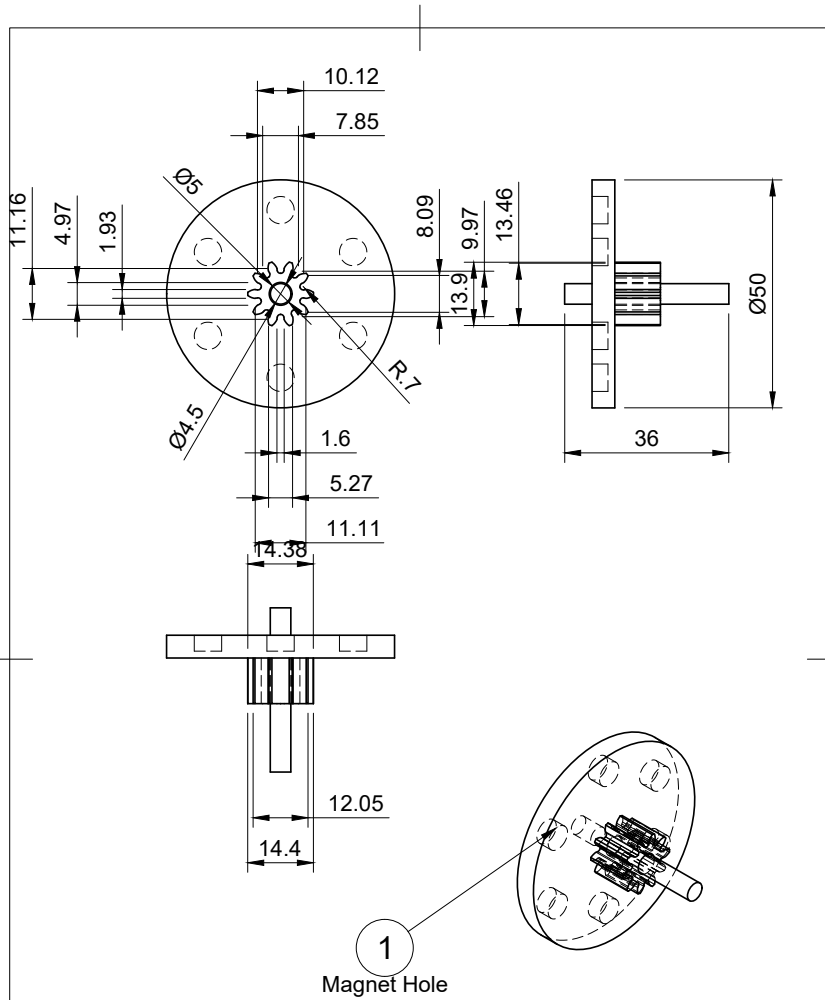


Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title Aft Prop House	DWG No.
		Rev.	Date of issue
			Sheet 14/30

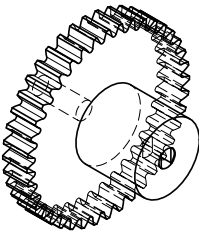




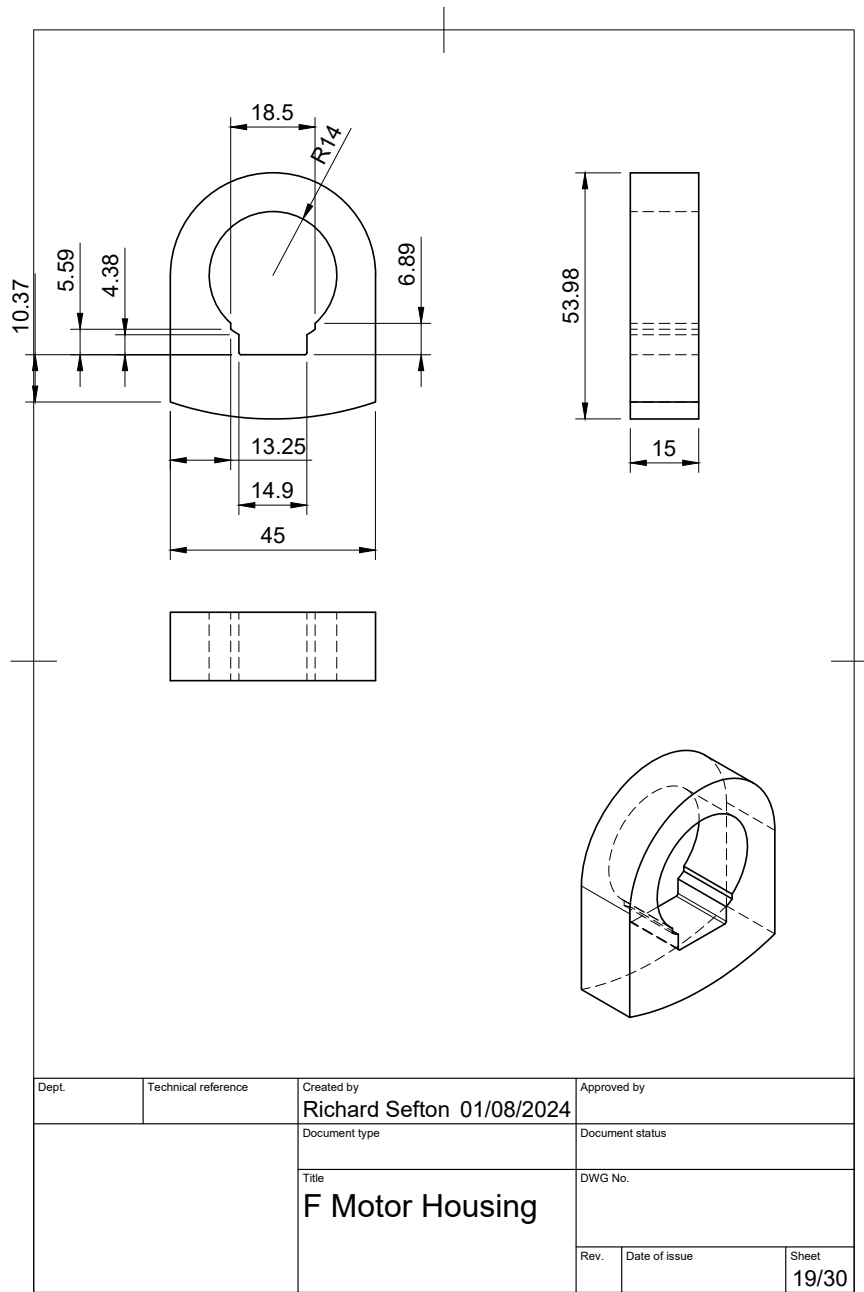
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title Main Prop	DWG No.
		Rev.	Date of issue
			Sheet 16/30

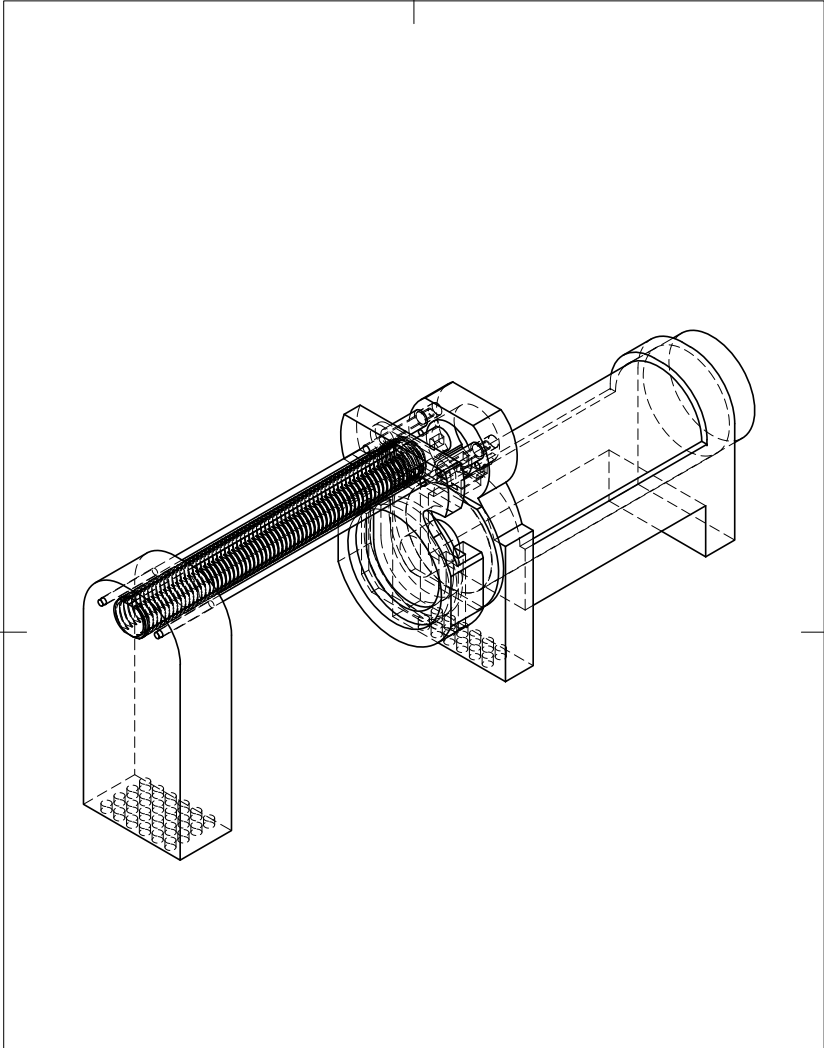


Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title F Prop Gear	DWG No.
		Rev.	Date of issue
			Sheet 17/30

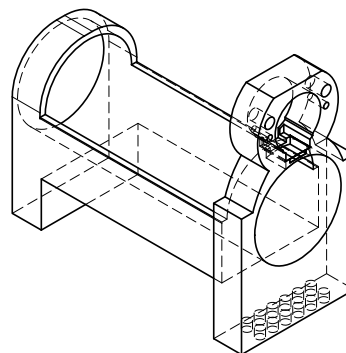
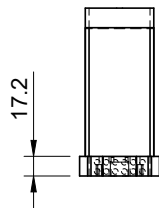


Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by		
		Document type	Document status		
		Title F Motor Gear	DWG No.		
			Rev.	Date of issue	Sheet 18/30

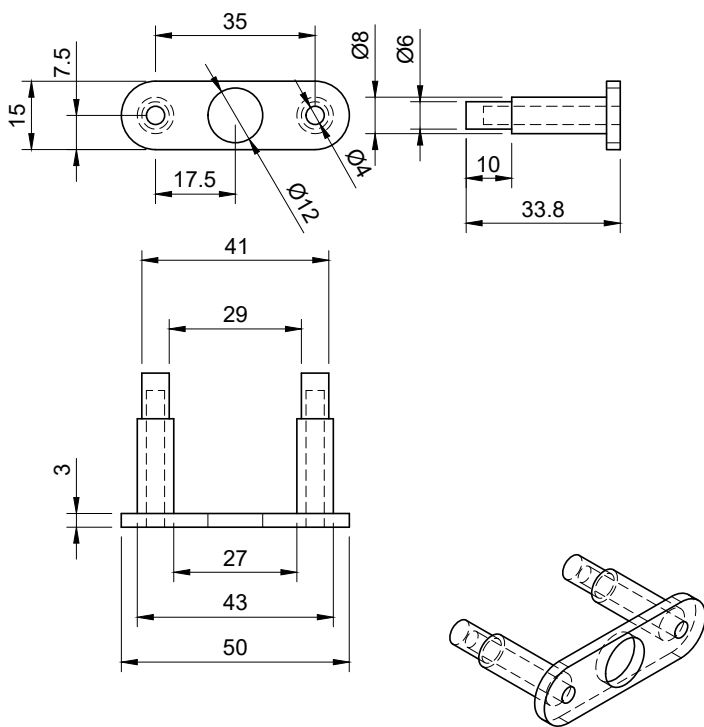




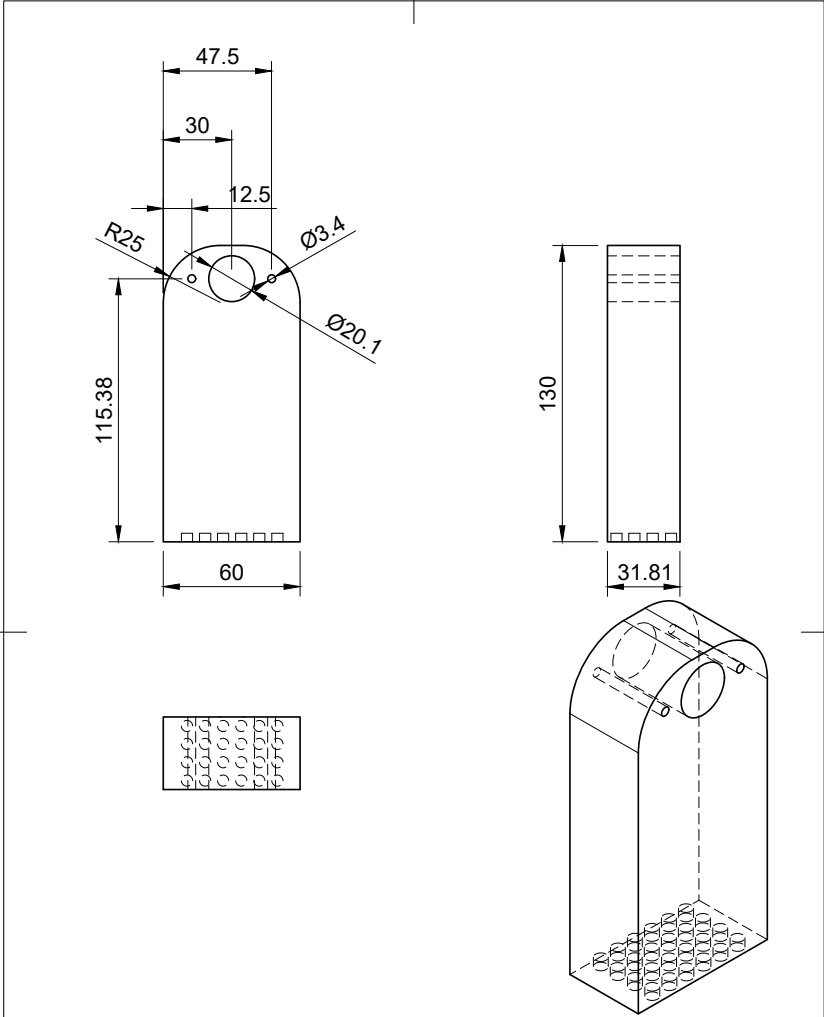
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by	
		Document type	Document status	
		Title Syringe Housing	DWG No.	
			Rev.	Date of issue
			Sheet	20/30



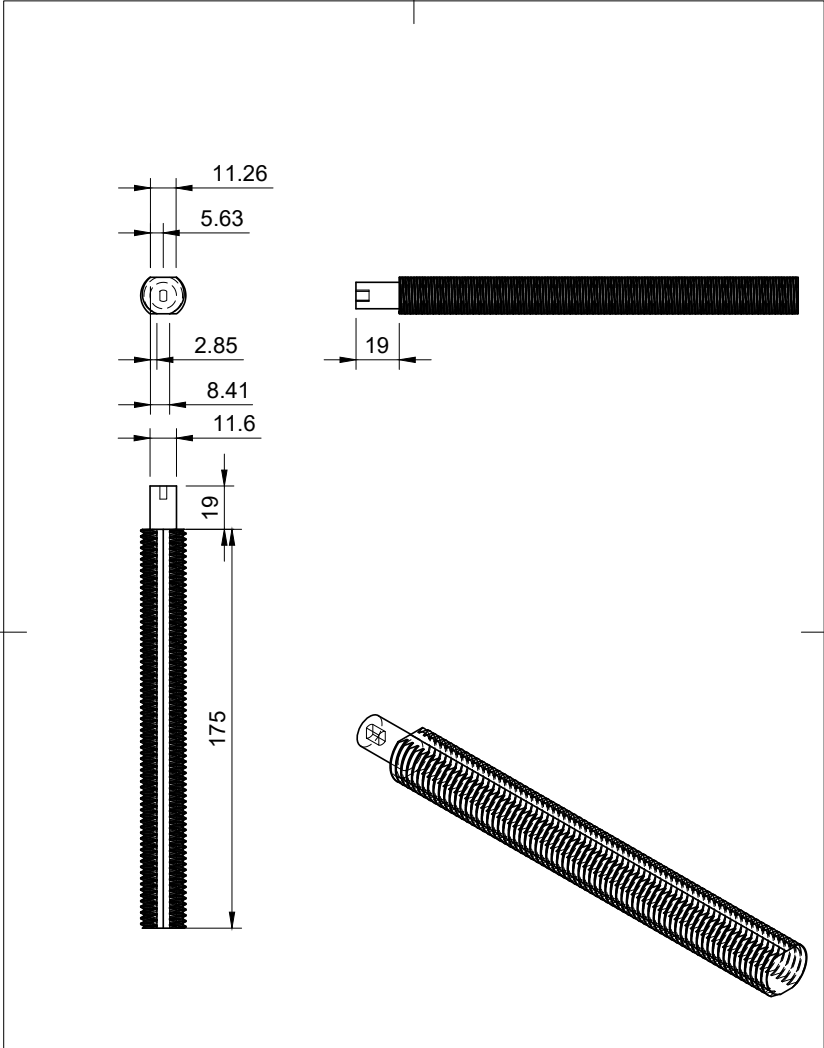
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by		
		Document type	Document status		
		Title Syringe Housing	DWG No.		
			Rev.	Date of issue	Sheet 21/30



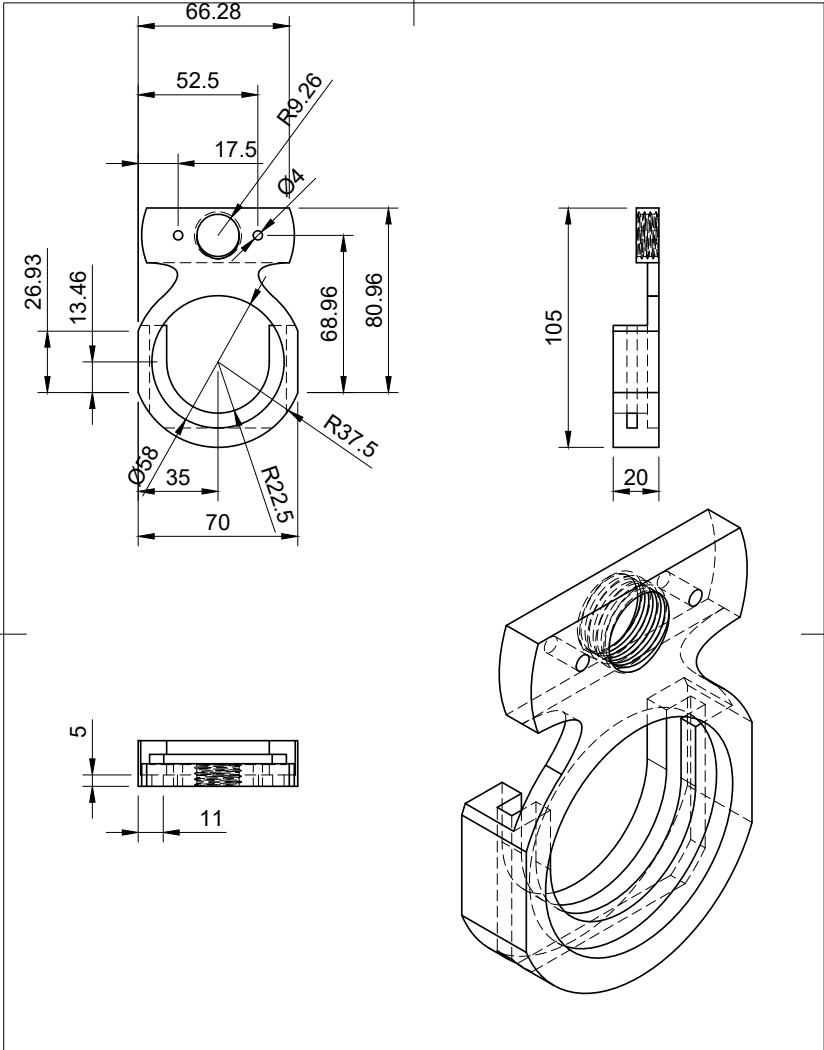
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title Shaft Holder	DWG No.
		Rev.	Date of issue
			Sheet 22/30



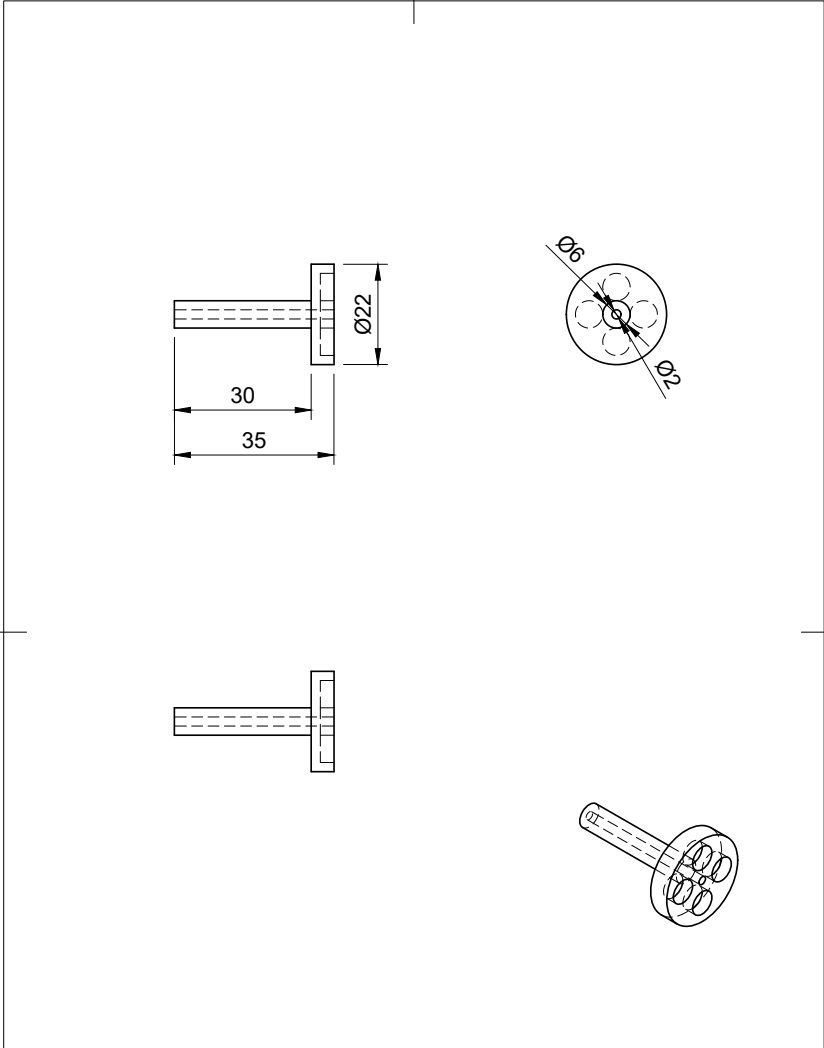
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title Shaft Stopper	DWG No.
		Rev.	Date of issue
		Sheet	23/30



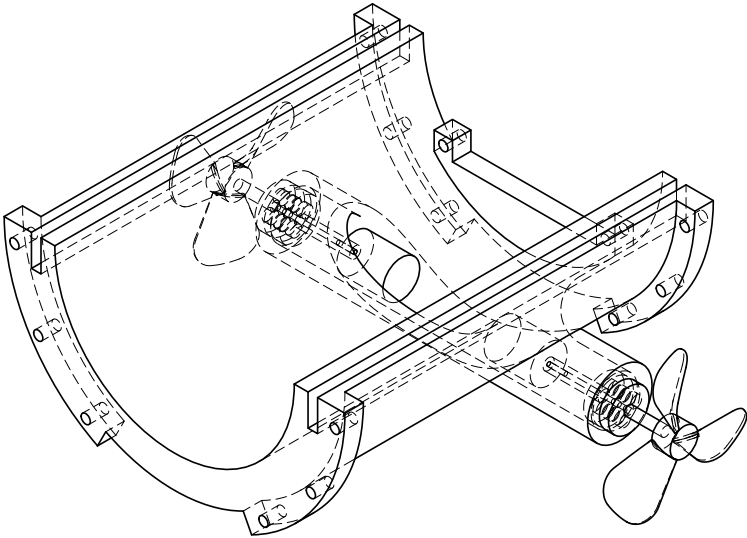
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by		
		Document type	Document status		
		Title Shaft	DWG No.		
			Rev.	Date of issue	Sheet 24/30



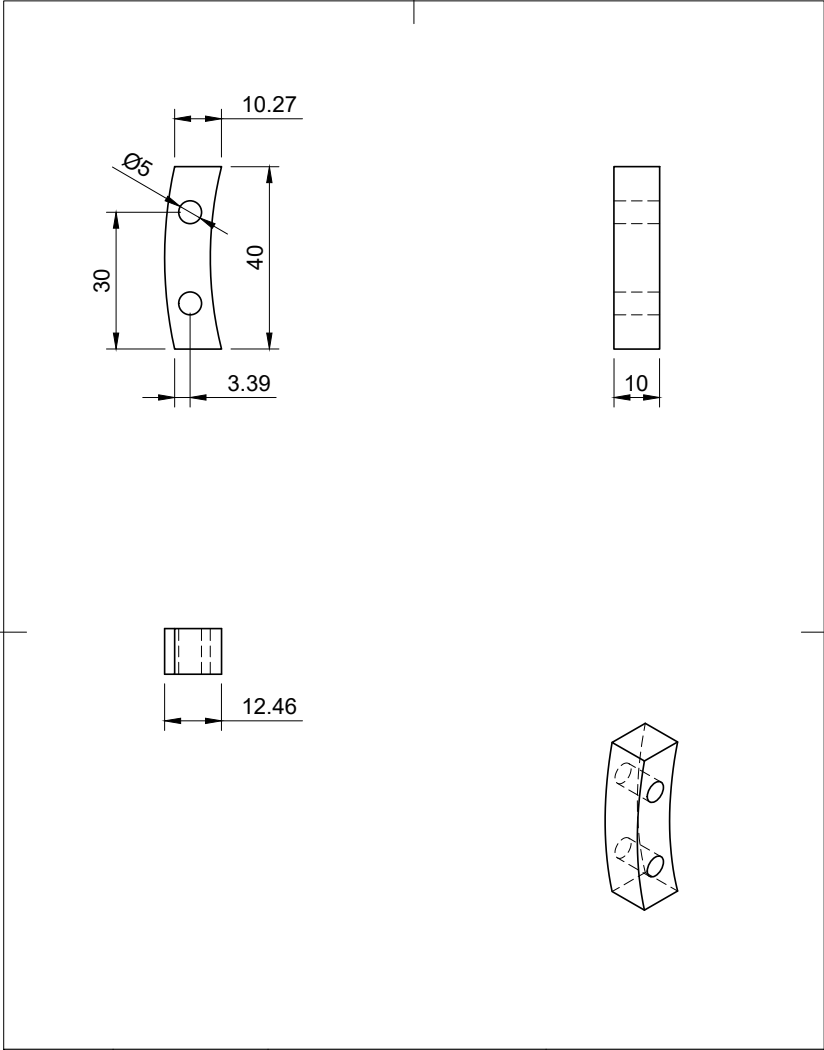
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by
		Document type	Document status
		Title Linear Nut	DWG No.
		Rev.	Date of issue
			Sheet 25/30



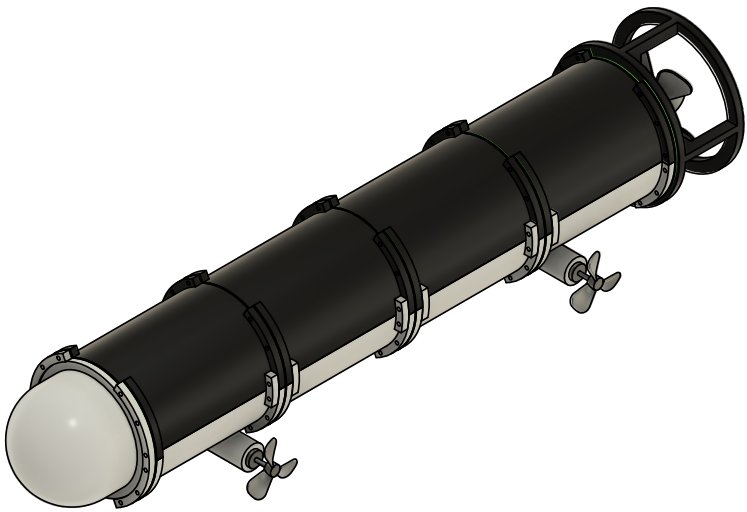
Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by	
		Document type	Document status	
		Title Side Shaft	DWG No.	
			Rev.	Date of issue
			Sheet	27/30



Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by	
		Document type	Document status	
		Title Side Prop Assmbly	DWG No.	
			Rev.	Date of issue
			Sheet	28/30



Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by	
		Document type	Document status	
		Title Fastner	DWG No.	
			Rev.	Date of issue
			Sheet	29/30



Dept.	Technical reference	Created by Richard Sefton 01/08/2024	Approved by		
		Document type	Document status		
		Title AUV Modelled	DWG No.		
			Rev.	Date of issue	Sheet 30/30

1.8 Final Code

1.8.1 Common ShortTypes.h

```
1      #ifndef SHORTTYPES_H
2      #define SHORTTYPES_H
3
4      #include<stdint.h>
5      #include <stdbool.h>
6
7      typedef uint8_t u8;
8      typedef int8_t i8;
9
10     typedef uint16_t u16;
11     typedef int16_t i16;
12
13     typedef uint32_t u32;
14     typedef int32_t i32;
15
16     #endif  /* SHORTTYPES_H */
```

1.8.2 Common Common.h

```
1      #ifndef COMMON_H
2      #define COMMON_H
3
4      #include <avr/io.h>
5      #include "ShortTypes.h"
6
7      extern enum {
8          RED = PIN6_bm,
9          GREEN = PIN5_bm,
10         BLUE = PIN4_bm
11     } Colours;
12
13     void setupRGB(void);
14     void RGB(u8);
15
16
17     #endif  /* COMMON_H */
```

1.8.3 Common Common.c

```
1      #include "Common.h"
2
```

```

3     void setupRGB(void) {
4         PORTB.DIR |= RED | GREEN | BLUE;
5         PORTB.OUT |= (RED | GREEN | BLUE);
6     }
7
8     void RGB(u8 colour) {
9         PORTB.OUT |= RED | GREEN | BLUE;
10        PORTB.OUT &= ~colour;
11    }

```

1.8.4 Central Controller AddressBook.h

```

1     #ifndef ADDRESSBOOK_H
2     #define ADDRESSBOOK_H
3
4     #define DEPTH_CONTROLLER 0x40
5     #define US_BOTTOM 0x30
6     #define US_RIGHT 0x31
7     #define US_LEFT 0x32
8     #define US_FORWARD 0x33
9     #define FORWARD_MOTOR 0x45
10    #define LEFT_MOTOR 0x46
11    #define RIGHT_MOTOR 0x47
12
13    #endif /* ADDRESSBOOK_H */

```

1.8.5 Central Controller AUV.h

```

1     #ifndef MODULES_H
2     #define MODULES_H
3
4     #include "ShortTypes.h"
5
6     typedef enum {
7         FMD_FORWARD = 1,
8         FMD_STOP = 2,
9         FMD_BACKWARD = 3
10    } ForwardMotorDirection;
11
12    typedef enum {
13        SMD_NONE = 0,
14        SMD_LEFT = 1,
15        SMD_RIGHT = 2
16    } SideMotorDirection;
17
18    typedef enum {

```

```

19         SMS_ON = 1,
20         SMS_OFF = 2
21     } SideMotorState;
22
23     typedef enum {
24         SD_NONE = 0,
25         SD_LOWER = 1,
26         SD_FORWARD = 2,
27         SD_LEFT = 3,
28         SD_RIGHT = 4
29     } SonarDirection;
30
31     typedef enum {
32         LOWER = 0,
33         FORWARD = 1,
34         LEFT = 2,
35         RIGHT = 3
36     } SonarModule;
37
38     typedef struct {
39         u16 distance;
40         SonarDirection direction;
41     } Sonar;
42
43     typedef enum {
44         LAND = 1,
45         WATER = 2,
46     } SonarMode;
47
48     typedef struct {
49         Sonar sonars [4];
50         ForwardMotorDirection mainMotorDirection;
51         SonarDirection mainMotorHeldBy;
52         SideMotorDirection sideMotorDirection;
53         SonarDirection sideMotorHeldBy;
54         u8 sonarIndex;
55         SonarMode sonarMode;
56         u8 secondsSubmerged;
57         u8 submergeTimer;
58         u8 secondsSurfaced;
59         u8 surfaceTimer;
60     } AUV;
61
62     void AUV_init(AUV*);
63     Sonar* AUV_loadSonar(AUV*, SonarModule);
64     void AUV_incrementSonarIndex(AUV*);
65     void AUV_setMotorDirection(AUV*, ForwardMotorDirection,
        SonarDirection);

```

```

66     void AUV_setTurnDirection(AUV*, SideMotorState,
        SonarDirection);
67     void AUV_setSonarMode(AUV*);
68     void AUV_increaseSubmergeTimer(AUV*);
69     u8 AUV_shouldSurface(AUV*);
70     void AUV_increaseSurfaceTimer(AUV*);
71     u8 AUV_shouldDive(AUV*);
72
73     #endif    /* MODULES_H */

```

1.8.6 Central Controller AUV.c

```

1
2     #include "AUV.h"
3     #include <avr/io.h>
4
5     void AUV_init(AUV* self) {
6         Sonar s1, s2, s3, s4;
7         s1.direction = SD_LOWER;
8         s2.direction = SD_FORWARD;
9         s3.direction = SD_LEFT;
10        s4.direction = SD_RIGHT;
11        self->sonars[0] = s1;
12        self->sonars[1] = s2;
13        self->sonars[2] = s3;
14        self->sonars[3] = s4;
15        self->sonarIndex = 0;
16        self->mainMotorDirection = FMD_STOP;
17        self->mainMotorHeldBy = SD_NONE;
18        self->sideMotorDirection = SMD_NONE;
19        self->sideMotorHeldBy = SD_NONE;
20        self->submergeTimer = 120;
21        self->surfaceTimer = 30;
22
23        //Using pins for mode detection. Votage will mean we're in
           water mode.
24        //PA5(Pin 6) Will be the voltage (OUTPUT)
25        PORTA.DIR |= PIN5_bm;
26        PORTA.OUT |= PIN5_bm;
27        //PA7(Pin 8) will be the detection pin. If high, we're in
           Water mode.
28        PORTA.DIR &= ~(PIN7_bm);
29    }
30
31    Sonar* AUV_loadSonar(AUV* self, SonarModule module) {
32        return &self->sonars[module];
33    }

```



```

34
35 void AUV_incrementSonarIndex(AUV* self) {
36     self->sonarIndex++;
37     if (self->sonarIndex > 3) {
38         self->sonarIndex = 0;
39     }
40 }
41
42 void AUV_setMotorDirection(AUV* self, ForwardMotorDirection
    dir, SonarDirection hold) {
43     self->mainMotorDirection = dir;
44     self->mainMotorHeldBy = hold;
45 }
46
47 void AUV_setTurnDirection(AUV* self, SideMotorState dir,
    SonarDirection hold) {
48     self->sideMotorDirection = dir;
49     self->sideMotorHeldBy = hold;
50 }
51
52 void AUV_setSonarMode(AUV* self) {
53     //Set to land mode.
54     self->sonarMode = LAND;
55     //If theres voltage on the detection pin, we're in water
    mode.
56     if (PORTA.IN & PIN7_bm) {
57         self->sonarMode = WATER;
58     }
59 }
60
61 void AUV_increaseSubmergeTimer(AUV* self) {
62     self->secondsSubmerged++;
63 }
64
65 u8 AUV_shouldSurface(AUV* self) {
66     if (self->secondsSubmerged >= self->submergeTimer) {
67         self->secondsSubmerged = 0;
68         return 1;
69     }
70     return 0;
71 }
72
73 void AUV_increaseSurfaceTimer(AUV* self) {
74     self->secondsSurfaced++;
75 }
76
77 u8 AUV_shouldDive(AUV* self) {
78     if (self->secondsSurfaced >= self->surfaceTimer) {
79         self->secondsSurfaced = 0;

```

```

80         return 1;
81     }
82     return 0;
83 }

```

1.8.7 Central Controller CentralController.h

```

1     #define F_CPU 333333UL
2
3     #include <avr/io.h>
4     #include <util/delay.h>
5     #include <avr/interrupt.h>
6     #include "CWire.h"
7     #include "ShortTypes.h"
8     #include "AddressBook.h"
9     #include "AUV.h"
10    #include "Common.h"
11
12    u8 depth = 0;
13
14    void setup(void);
15    void mainClkCtrl(void);
16    void setupRTC(void);
17    u16 ping(SonarModule);
18    void handleDistanceResponse(SonarDirection);
19    u8 getDepth(void);
20    void dive(u8);
21    void raise(u8);
22    void commandForwardMotor(ForwardMotorDirection);
23    void commandSideMotors(SideMotorDirection);
24
25    TwoWire twi0;
26    AUV auv;
27
28    int main() {
29        setup();
30
31        //Give other modules time to init
32        RGB(RED);
33        _delay_ms(2000);
34
35        TwoWire_init(&twi0, &TWI0);
36        TwoWire_Master_begin(&twi0);
37
38        //Wait for depth controller to enter home position.
39        while(getDepth() != 1) {
40            RGB(RED);

```

```

41         _delay_ms(1000);
42     }
43
44     _delay_ms(2000);
45
46     RGB(GREEN);
47
48     //DIVE, DIVE, DIVE!
49     dive(0xFF);
50
51     sei();
52
53     while(1) {
54
55     }
56
57     return 0;
58 }
59
60 void setup(void) {
61     mainClkCtrl();
62     setupRTC();
63     setupRGB();
64     AUV_init(&auv);
65 }
66
67 void mainClkCtrl(void)
68 {
69     _PROTECTED_WRITE(CLKCTRL.MCLKCTRLA,
70                     CLKCTRL_CLKSEL_OSC20M_gc);
71     _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_6X_gc |
72                     CLKCTRL_PEN_bm);
73     // F_CPU with this configuration will be 3.33MHz
74 }
75
76 void setupRTC(void) {
77     RTC.CLKSEL = RTC_CLKSEL_INT1K_gc;
78     while(RTC.STATUS);
79     RTC.CTRLA |= RTC_PRESCALER_DIV1_gc;
80     RTC.PER = 1024;
81     while (RTC.STATUS);
82     RTC.INTFLAGS |= RTC_OVF_bm;
83     RTC.INTCTRL |= RTC_OVF_bm;
84     while (RTC.STATUS);
85     RTC.CTRLA |= RTC_RTCEN_bm;
86     while (RTC.STATUS);
87 }
88
89 u16 ping(SonarModule module) {

```

```

88     AUV_setSonarMode(&auv);
89     //initiate sonar
90     u8 addr = 0x00;
91     switch(module) {
92         case LOWER: {
93             addr = US_BOTTOM;
94             break;
95         }
96         case FORWARD: {
97             addr = US_FORWARD;
98             break;
99         }
100        case LEFT: {
101            addr = US_LEFT;
102            break;
103        }
104        case RIGHT: {
105            addr = US_RIGHT;
106            break;
107        }
108        default: break;
109    }
110    cli();
111    TwoWire_beginTransmission(&twi0, addr);
112    TwoWire_write(&twi0, auv.sonarMode); //The value is the
        mode of operation
113    TwoWire_endTransmission(&twi0, 1);
114    _delay_ms(250); //Wait before responding
115
116    //get the results
117    u16 dist = 0;
118    TwoWire_requestFrom(&twi0, addr, 2, 1);
119    if (TwoWire_available(&twi0) == 2) {
120        u8 dataLow = TwoWire_read(&twi0);
121        u8 dataHigh = TwoWire_read(&twi0);
122        dist = dataLow;
123        dist |= (dataHigh << 8);
124    }
125    sei();
126
127    //Becuase the sensor is mounted on the bottom, we can't
        actively test this now its
128    //assembled. It was tested as working previously but now we
        need to override the
129    //returned value.
130    if ((addr == US_BOTTOM) && (auv.sonarMode == LAND)) {
131        return 0xFFFF; //Return max.
132    }
133    return dist;

```

```

134     }
135
136 void handleDistanceResponse(SonarDirection dir) {
137     switch (dir) {
138         case SD_LOWER: {
139             Sonar* s = AUV_loadSonar(&auv, LOWER);
140             if (s->distance > 0 && s->distance < 1400) {
141                 depth = getDepth();
142                 if (depth == 0) {
143                     //Handle later
144                 } else {
145                     /**
146                     * I think this is having the same ghandi bug
147                     * if depth is less than 10 and we take off 10 it
148                     * does to 255 - depth - 10
149                     * If its less than 10 we just need to raise to 0.
150                     */
151                     if (depth > 10) {
152                         depth -= 10;
153                         raise(depth);
154                     } else if (depth == 1) {
155                         raise(1); //not doing anything in reality.
156                     } else {
157                         raise(0); //raise to 0 and it should 1 itself.
158                     }
159                 }
160                 AUV_setMotorDirection(&auv, FMD_STOP, SD_LOWER);
161             } else if (s->distance > 0 && s->distance > 3000) {
162                 // if (depth < 245) {
163                 //     depth += 10;
164                 //     dive(depth);
165                 // } else if (depth == 254) {
166                 //     dive(254);
167                 // } else {
168                 //     dive(255);
169                 // }
170                 if (auv.mainMotorHeldBy == SD_LOWER ||
171                     auv.mainMotorHeldBy == SD_NONE) {
172                     AUV_setMotorDirection(&auv, FMD_FORWARD,
173                         SD_NONE);
174                 }
175             }
176         }
177     }
178
179     case SD_FORWARD: {
180         Sonar* s = AUV_loadSonar(&auv, FORWARD);
181         if (s->distance > 0 && s->distance < 1400) {

```

```

179         if(s->distance < 800) {
180             AUV_setMotorDirection(&auv,FMD_BACKWARD,
                SD_FORWARD);
181         } else {
182             AUV_setMotorDirection(&auv, FMD_STOP,
                SD_FORWARD);
183         }
184     } else if(s->distance > 0 && s->distance > 1400) {
185         if (auv.mainMotorHeldBy == SD_FORWARD ||
            auv.mainMotorHeldBy == SD_NONE) {
186             AUV_setMotorDirection(&auv, FMD_FORWARD,
                SD_NONE);
187         }
188     }
189 }
190
191 case SD_LEFT: {
192     Sonar* s = AUV_loadSonar(&auv, LEFT);
193     if (s->distance < 1400) {
194         AUV_setMotorDirection(&auv,FMD_STOP, SD_LEFT);
195         AUV_setTurnDirection(&auv, SMD_RIGHT, SD_LEFT);
196     } else {
197         if (auv.mainMotorHeldBy == SD_LEFT ||
            auv.mainMotorHeldBy == SD_NONE) {
198             AUV_setMotorDirection(&auv, FMD_FORWARD,
                SD_NONE);
199         }
200         if (auv.sideMotorHeldBy == SD_LEFT ||
            auv.sideMotorHeldBy == SD_NONE) {
201             AUV_setTurnDirection(&auv, SMD_NONE, SD_NONE);
202         }
203     }
204 }
205
206 case SD_RIGHT: {
207     Sonar* s = AUV_loadSonar(&auv, LEFT);
208     if (s->distance < 800) {
209         AUV_setMotorDirection(&auv,FMD_STOP, SD_RIGHT);
210         AUV_setTurnDirection(&auv, SMD_LEFT, SD_RIGHT);
211     } else {
212         if (auv.mainMotorHeldBy == SD_RIGHT ||
            auv.mainMotorHeldBy == SD_NONE) {
213             AUV_setMotorDirection(&auv, FMD_FORWARD,
                SD_NONE);
214         }
215         if (auv.sideMotorHeldBy == SD_RIGHT ||
            auv.sideMotorHeldBy == SD_NONE) {
216             AUV_setTurnDirection(&auv, SMD_NONE, SD_NONE);
217         }

```

```

218         }
219     }
220
221     default: {
222         break;
223     }
224 }
225 }
226
227 u8 getDepth(void) {
228     RGB(BLUE);
229     _delay_ms(100); //So we can see the xmission
230     u8 data = 0;
231     TwoWire_requestFrom(&twi0, DEPTH_CONTROLLER, 1, 1);
232     if (TwoWire_available(&twi0)) {
233         data = TwoWire_read(&twi0); // Read the received byte
234     }
235     return data;
236 }
237
238 void dive(u8 d) {
239     RGB(BLUE);
240     TwoWire_beginTransmission(&twi0, DEPTH_CONTROLLER);
241     TwoWire_write(&twi0, d);
242     TwoWire_endTransmission(&twi0, 1);
243     RGB(GREEN);
244 }
245 void raise(u8 d) {
246     dive(d);
247 }
248
249 void commandForwardMotor(ForwardMotorDirection dir) {
250     RGB(BLUE);
251     TwoWire_beginTransmission(&twi0, FORWARD_MOTOR);
252     TwoWire_write(&twi0, (u8)dir);
253     TwoWire_endTransmission(&twi0, 1);
254     RGB(GREEN);
255 }
256
257 void commandSideMotors(SideMotorDirection dir) {
258     RGB(BLUE);
259     if (dir == SMD_LEFT) {
260         TwoWire_beginTransmission(&twi0, LEFT_MOTOR);
261         TwoWire_write(&twi0, (u8)SMS_OFF);
262         TwoWire_endTransmission(&twi0, 1);
263         _delay_ms(100);
264         TwoWire_beginTransmission(&twi0, RIGHT_MOTOR);
265         TwoWire_write(&twi0, (u8)SMS_ON);
266         TwoWire_endTransmission(&twi0, 1);

```

```

267     } else if (dir == SMD_RIGHT) {
268         TwoWire_beginTransmission(&twi0, RIGHT_MOTOR);
269         TwoWire_write(&twi0, (u8)SMS_OFF);
270         TwoWire_endTransmission(&twi0, 1);
271         _delay_ms(100);
272         TwoWire_beginTransmission(&twi0, LEFT_MOTOR);
273         TwoWire_write(&twi0, (u8)SMS_ON);
274         TwoWire_endTransmission(&twi0, 1);
275     } else if (dir == SMD_NONE) {
276         TwoWire_beginTransmission(&twi0, RIGHT_MOTOR);
277         TwoWire_write(&twi0, (u8)SMS_OFF);
278         TwoWire_endTransmission(&twi0, 1);
279         _delay_ms(100);
280         TwoWire_beginTransmission(&twi0, LEFT_MOTOR);
281         TwoWire_write(&twi0, (u8)SMS_OFF);
282         TwoWire_endTransmission(&twi0, 1);
283     }
284     RGB(GREEN);
285 }
286
287 ISR(RTC_CNT_vect) {
288     RGB(BLUE);
289     switch(auv.sonarIndex) {
290         case LOWER: {
291             auv.sonars[LOWER].distance = ping(LOWER);
292             RGB(GREEN);
293             handleDistanceResponse(SD_LOWER);
294             break;
295         }
296         case FORWARD: {
297             auv.sonars[FORWARD].distance = ping(FORWARD);
298             RGB(GREEN);
299             handleDistanceResponse(SD_FORWARD);
300             break;
301         }
302         case LEFT: {
303             auv.sonars[LEFT].distance = ping(LEFT);
304             RGB(GREEN);
305             handleDistanceResponse(SD_LEFT);
306             break;
307         }
308         case RIGHT: {
309             auv.sonars[RIGHT].distance = ping(RIGHT);
310             RGB(GREEN);
311             handleDistanceResponse(SD_RIGHT);
312             break;
313         }
314         default: break;
315     }

```



```

316     AUV_incrementSonarIndex(&auv);
317     commandForwardMotor(auv.mainMotorDirection);
318     commandSideMotors(auv.sideMotorDirection);
319
320     if ((depth == 0) || (depth == 1)) {
321         AUV_increaseSurfaceTimer(&auv);
322         if (AUV_shouldDive(&auv) == 1) {
323             dive(255);
324         }
325     } else {
326         AUV_increaseSubmergeTimer(&auv);
327         if (AUV_shouldSurface(&auv)) {
328             raise(0);
329         }
330     }
331
332     RTC.INTFLAGS = RTC_OVF_bm;
333 }
334

```

1.8.8 Stepper Motor StepperMotor.h

```

1     #ifndef STEPPERMOTOR_H
2     #define STEPPERMOTOR_H
3
4     #ifndef F_CPU
5     #define F_CPU 3333333UL
6     #endif
7
8     #include <avr/io.h>
9     #include <util/delay.h>
10    #include "Common.h"
11
12    /*
13    * Lets apply the pins to some defines
14    *
15    *
16    *
17    * StepperMotor
18    * 17 - PC0
19    * 18 - PC1
20    * 19 - PC2
21    * 20 - PC3
22    *
23    * Going to use 6(PA5), 7(PA6). These need to be inputs to
      detect voltage

```

```

24      * Will also need interrupts on these and pullup enabled
      * because they're floating
25      *
26      * Button we'll put on PB2
27      * Also we'll add the pullup because while theres no voltage
      * its also floating.
28      */
29      extern enum {
30          STEP_PIN_1 = PIN0_bm,
31          STEP_PIN_2 = PIN1_bm,
32          STEP_PIN_3 = PIN2_bm,
33          STEP_PIN_4 = PIN3_bm
34      } StepperPins;
35
36      typedef struct {
37          i8 step;
38      } StepperMotor;
39
40      void StepperMotor_init(StepperMotor*);
41      void StepperMotor_setup(void);
42      void StepperMotor_step(StepperMotor*);
43      void StepperMotor_increaseStep(StepperMotor*);
44      void StepperMotor_decreaseStep(StepperMotor*);
45      void StepperMotor_allStop(void);
46
47      #endif /* STEPPERMOTOR_H */

```

1.8.9 Stepper Motor StepperMotor.c

```

1      #include "StepperMotor.h"
2
3      void StepperMotor_init(StepperMotor* self) {
4          StepperMotor_setup();
5          self->step = 0;
6      }
7
8      void StepperMotor_setup(void) {
9          PORTC.DIR |= STEP_PIN_1 | STEP_PIN_2 | STEP_PIN_3 |
          STEP_PIN_4;
10     }
11
12     void StepperMotor_step(StepperMotor* self) {
13         switch(self->step) {
14             case 0:
15                 PORTC.OUTCLR |= STEP_PIN_1 | STEP_PIN_2 | STEP_PIN_3;
16                 PORTC.OUTSET |= STEP_PIN_4;
17                 break;

```

```

18         case 1:
19             PORTC.OUTCLR |= STEP_PIN_1 | STEP_PIN_2;
20             PORTC.OUTSET |= STEP_PIN_3 | STEP_PIN_4;
21             break;
22         case 2:
23             PORTC.OUTCLR |= STEP_PIN_1 | STEP_PIN_2 | STEP_PIN_4;
24             PORTC.OUTSET |= STEP_PIN_3;
25             break;
26         case 3:
27             PORTC.OUTCLR |= STEP_PIN_1 | STEP_PIN_4;
28             PORTC.OUTSET |= STEP_PIN_2 | STEP_PIN_3;
29             break;
30         case 4:
31             PORTC.OUTCLR |= STEP_PIN_1 | STEP_PIN_3 | STEP_PIN_4;
32             PORTC.OUTSET |= STEP_PIN_2;
33             break;
34         case 5:
35             PORTC.OUTCLR |= STEP_PIN_3 | STEP_PIN_4;
36             PORTC.OUTSET |= STEP_PIN_1 | STEP_PIN_2;
37             break;
38         case 6:
39             PORTC.OUTCLR |= STEP_PIN_2 | STEP_PIN_3 | STEP_PIN_4;
40             PORTC.OUTSET |= STEP_PIN_1;
41             break;
42         case 7:
43             PORTC.OUTCLR |= STEP_PIN_2 | STEP_PIN_3;
44             PORTC.OUTSET |= STEP_PIN_1 | STEP_PIN_4;
45             break;
46         default:
47             break;
48     }
49 }
50
51 void StepperMotor_increaseStep(StepperMotor* self) {
52     self->step++;
53     if (self->step > 7) {
54         self->step = 0;
55     }
56 }
57
58 void StepperMotor_decreaseStep(StepperMotor* self) {
59     self->step--;
60     if (self->step < 0) {
61         self->step = 7;
62     }
63 }
64
65 void StepperMotor_allStop(void) {

```

```

66         PORTC.OUTCLR |= STEP_PIN_1 | STEP_PIN_2 | STEP_PIN_3 |
        STEP_PIN_4;
67     }

```

1.8.10 Depth Controller DepthController.h

```

1     #define F_CPU 3333333UL
2     #include <avr/io.h>
3     #include <util/delay.h>
4     #include <avr/interrupt.h>
5     #include "CWire.h"
6     #include "ShortTypes.h"
7     #include "Common.h"
8     #include "StepperMotor.h"
9
10    u8 plungerPos = 255;
11    u8 commandedPos = 0;
12
13    #define THREAD_OUT 0x01
14    #define THREAD_IN 0x02
15    #define STOP 0x00
16    #define GO 0x01
17
18    u8 dir = THREAD_IN;
19    u8 run = GO;
20
21    //PortA
22    #define BUFFER_OUT_PIN PIN5_bm
23    #define BUFFER_IN_PIN PIN7_bm
24
25    //Functions we need to define.
26    //This isn't in pseudocode. I like to include it to be explicit
27    void mainClkCtrl(void);
28    void setup(void);
29    void setupPins(void);
30    void setupRTC(void);
31
32    //My TWI Library requires callbacks.
33    void I2C_RX_Callback(u8);
34    void I2C_TX_Callback(void);
35    //Also need an address we can bind this module to.
36
37    TwoWire twi0;
38    StepperMotor stepper;
39
40    #define ADDR 0x40
41

```

```

42     int main(void) {
43         setup();
44
45         TwoWire_init(&twi0, &TWI0);
46         TwoWire_Slave_begin(&twi0, ADDR, 0, 0);
47
48         TwoWire_onReceive(&twi0, I2C_RX_Callback);
49         TwoWire_onRequest(&twi0, I2C_TX_Callback);
50
51         sei();
52
53         while(1) {
54             if (plungerPos != commandedPos) {
55                 run = GO;
56                 RGB(BLUE);
57                 if ((dir == THREAD_OUT && plungerPos != 254) &&
58                     plungerPos != commandedPos){
59                     StepperMotor_step(&stepper);
60                     StepperMotor_decreaseStep(&stepper);
61                 } else if ((dir == THREAD_IN && plungerPos != 1) &&
62                     plungerPos != commandedPos) {
63                     StepperMotor_step(&stepper);
64                     StepperMotor_increaseStep(&stepper);
65                 }
66                 _delay_us(750);
67             } else {
68                 run = STOP;
69                 RGB(GREEN);
70                 StepperMotor_allStop();
71             }
72         }
73
74         return 0;
75     }
76
77     void mainClkCtrl(void)
78     {
79         _PROTECTED_WRITE(CLKCTRL.MCLKCTRLA,
80             CLKCTRL_CLKSEL_OSC20M_gc | CLKCTRL_CLKOUT_bm);
81         _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_6X_gc |
82             CLKCTRL_PEN_bm);
83     }
84
85     void setup(void) {
86         mainClkCtrl();
87         setupRTC();
88         setupPins();
89         StepperMotor_init(&stepper);
90         setupRGB();

```

```

87     }
88
89     void setupRTC(void) {
90         RTC.CLKSEL = RTC_CLKSEL_INT1K_gc;
91         while(RTC.STATUS);
92         RTC.CTRLA |= RTC_PRESCALER_DIV1_gc;
93         /**
94          * End to end takes 2min 25s/145s
95          *
96          * Going to knock off 5s so we should be clear of the buffer.
97          *
98          *  $145/255 = 0.57$  and change. So one tick of the pos is
99             worth  $\sim 0.57s$ 
100          */
101         RTC.PER = 582; //Needed to add 12 as 1k is 1024 kHz
102         while (RTC.STATUS);
103         RTC.INTFLAGS |= RTC_OVF_bm;
104         RTC.INTCTRL |= RTC_OVF_bm;
105         while (RTC.STATUS);
106         RTC.CTRLA |= RTC_RTCEN_bm;
107         while (RTC.STATUS);
108     }
109
110     void setupPins(void) {
111         //Buffers
112         PORTA.DIR &= ~(BUFFER_OUT_PIN);
113         PORTA.DIR &= ~(BUFFER_IN_PIN);
114
115         //This pin is for the voltage to be detected.
116         PORTA.DIR |= PIN4_bm;
117         PORTA.OUT |= PIN4_bm;
118
119         PORTA.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_RISING_gc;
120         PORTA.PIN7CTRL |= PORT_PULLUPEN_bm | PORT_ISC_RISING_gc;
121     }
122
123     void I2C_RX_Callback(u8 nBytes) {
124         u8 data = 0;
125         while (TwoWire_available(&twi0)) {
126             data = TwoWire_read(&twi0);
127         }
128
129         if (data > plungerPos) {
130             dir = THREAD_OUT;
131         } else {
132             dir = THREAD_IN;
133         }
134
135         commandedPos = data;

```

```

135     }
136
137     void I2C_TX_Callback(void) {
138         //This is used in read requests where the controller is
139         expecting the depth.
140
141         //without getting the current depth we would risk bottoming
142         out particularly
143         //in the first call to dive which sends to max depth.
144         TwoWire_write(&twi0, plungerPos);
145     }
146
147     //ISRS
148     ISR(RTC_CNT_vect) {
149         RTC.INTFLAGS = RTC_OVF_bm;
150         if (run == GO) {
151             if (dir == THREAD_OUT) {
152                 plungerPos += 1;
153             } else if (dir == THREAD_IN) {
154                 plungerPos -= 1;
155             }
156         }
157         //Incase we naturally get there without using the buffer.
158         We don't want to sit at 0
159         // if (plungerPos == 1) {
160         //     commandedPos = 1;
161         //     dir = THREAD_IN;
162         // } else if (plungerPos == 254 && commandedPos > 0) {
163         //     commandedPos = 254;
164         //     dir = THREAD_OUT;
165         // }
166     }
167
168     ISR(PORTA_PORT_vect) {
169         if (PORTA.INTFLAGS & BUFFER_OUT_PIN) {
170             if (!(PORTA.IN & BUFFER_OUT_PIN)) {
171                 RTC.CNT = 0;
172                 plungerPos = 255;
173                 commandedPos = (254);
174                 dir = THREAD_IN;
175             }
176             PORTA.INTFLAGS |= BUFFER_OUT_PIN;
177         }
178
179         if (PORTA.INTFLAGS & BUFFER_IN_PIN) {
180             if (!(PORTA.IN & BUFFER_IN_PIN)) {
181                 RTC.CNT = 0;
182                 plungerPos = 0;
183                 commandedPos = 1;

```

```

181         dir = THREAD_OUT;
182     }
183     PORTA.INTFLAGS |= BUFFER_IN_PIN;
184 }
185 PORTA.INTFLAGS |= BUFFER_IN_PIN | BUFFER_OUT_PIN;
186 }

```

1.8.11 Sonar Sonar.h

```

1  #ifndef SONAR_H
2  #define SONAR_H
3
4  /**
5   Because the Sonar Module is infact multiple modules lets make
6   it a library project
7   * and call it in when needed.
8   */
9
10 #ifndef F_CPU
11 #define F_CPU 3333333UL
12 #endif
13
14 #include <avr/io.h>
15 #include <util/delay.h>
16 #include "Common.h"
17
18 //typedef uint8_t u8;
19 //typedef uint16_t u16;
20
21 extern enum {
22     TRIGGER = PIN3_bm,
23     ECHO = PIN2_bm
24 } S_Pins;
25
26 typedef enum {
27     LAND = 1,
28     WATER = 2,
29 } Mode;
30
31 typedef struct {
32     u8 as_u8;
33     u16 as_u16;
34     float as_float;
35 } Distance;
36
37 typedef struct {
38     Mode mode;

```



```

38         u16 ticks;
39         float sos_land;
40         float sos_water;
41         Distance distance;
42     } Sonar;
43
44     void Sonar_init(Sonar*, Mode);
45     void Sonar_setupSonar(void);
46     void Sonar_setupTCA(void);
47     void Sonar_enableTCA(void);
48     void Sonar_disableTCA(void);
49     void Sonar_trigger(Sonar*);
50     void Sonar_calculateDistance(Sonar*);
51     void Sonar_convertDistances(Sonar*);
52
53     #endif  /* SONAR_H */

```

1.8.12 Sonar Sonar.c

```

1     #include "Sonar.h"
2
3     void Sonar_init(Sonar* self, Mode mode) {
4         Sonar_setupSonar();
5         Sonar_setupTCA();
6         self->sos_land = 34300.0f;
7         self->sos_water = 148000.0f;
8         self->mode = mode;
9     }
10
11     void Sonar_setupSonar(void) {
12         PORTC.DIR |= TRIGGER;
13         PORTC.DIR &= ~ECHO;
14
15         PORTC.PIN1CTRL |= PORT_PULLUPEN_bm;
16         PORTC.PIN2CTRL |= PORT_PULLUPEN_bm;
17     }
18
19     void Sonar_setupTCA(void) {
20         TCA0.SINGLE.CTRLA |= TCA_SINGLE_CLKSEL_DIV1_gc;
21         TCA0.SINGLE.CNT = 0;
22         TCA0.SINGLE.PER = 0xFFFF;
23     }
24
25     void Sonar_enableTCA(void) {
26         TCA0.SINGLE.CNT = 0;
27         TCA0.SINGLE.CTRLA |= TCA_SINGLE_ENABLE_bm;
28     }

```

```

29
30 void Sonar_disableTCA(void) {
31     TCA0.SINGLE.CTRLA &= ~(TCA_SINGLE_ENABLE_bm);
32 }
33
34 void Sonar_trigger(Sonar* self) {
35     PORTC.OUTSET |= TRIGGER;
36     _delay_us(10);
37     PORTC.OUTCLR |= TRIGGER;
38     //wait for echo to go high
39     while(!(PORTC.IN & ECHO));
40     Sonar_enableTCA();
41     //Wait for echo to go High
42     while(PORTC.IN & ECHO);
43     Sonar_disableTCA();
44     self->ticks = TCA0.SINGLE.CNT;
45     Sonar_calculateDistance(self);
46 }
47
48 void Sonar_calculateDistance(Sonar* self) {
49     float cpu = (float)F_CPU / 64.0f; //Think this was using a
        prescaler of 64?
50     float time = (float)self->ticks / cpu;
51     float sos = 0.0f;
52     if (self->mode == LAND) {
53         sos = self->sos_land;
54     } else {
55         sos = self->sos_water;
56     }
57     self->distance.as_float = time * sos / 2.0f;
58     Sonar_convertDistances(self);
59 }
60
61 void Sonar_convertDistances(Sonar* self) {
62     //Should take the whole number of the float
63     float fdist = self->distance.as_float;
64     u16 u16dist = (u16)fdist;
65     u8 u8dist = (u8)(u16dist / 100);
66
67     self->distance.as_u16 = u16dist;
68     self->distance.as_u8 = u8dist;
69 }

```

1.8.13 Sonar SonarModule.h

```

1     #include <stdint.h>
2     #define F_CPU 3333333UL

```

```

3
4     #include <avr/io.h>
5     #include <util/delay.h>
6     #include <avr/interrupt.h>
7     #include "ShortTypes.h"
8     #include "CWire.h"
9     #include "Common.h"
10    #include "Sonar.h"
11
12    void setup(void);
13    void mainClkCtrl(void);
14
15    //TWI Library requires callbacks.
16    void I2C_RX_Callback(u8);
17    void I2C_TX_Callback(void);
18    //Also need an address
19    #define ADDR 0x30
20
21    TwoWire twi0;
22    Sonar sonar;
23
24    int main(void) {
25        setup();
26        RGB(RED);
27
28        TwoWire_init(&twi0, &TWI0);
29        TwoWire_Slave_begin(&twi0, ADDR, 0, 0);
30
31        TwoWire_onReceive(&twi0, I2C_RX_Callback);
32        TwoWire_onRequest(&twi0, I2C_TX_Callback);
33
34        sei();
35
36        RGB(GREEN);
37
38        while(1) {
39            //Do nothing.
40        }
41    }
42
43    void setup(void) {
44        mainClkCtrl();
45        Sonar_init(&sonar, LAND);
46        setupRGB();
47    }
48
49    void mainClkCtrl(void) {
50        _PROTECTED_WRITE(CLKCTRL.MCLKCTRLA,
            CLKCTRL_CLKSEL_OSC20M_gc | CLKCTRL_CLKOUT_bm);

```

```

51     _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_6X_gc |
52         CLKCTRL_PEN_bm);
53 }
54 void I2C_RX_Callback(u8 val) {
55     /**
56      * So the i2c rail isn't waiting for a response and runs a
57      * risk of timing out:
58      *
59      * We are going to trigger the ping from the master with a
60      * write command.
61      *
62      * Once triggered the master will wait (a time in ms, maybe
63      * 200ms) then
64      * perform a read request to get the results.
65      */
66     if (val == 0x01) { //We're only expecting 1 byte
67         while (TwoWire_available(&twi0)) {
68             u8 data = TwoWire_read(&twi0);
69             //arbitrary value just to prevent against accidental
70             //triggers from noise on the line
71             if ((data == LAND) || (data == WATER)) {
72                 if (data == LAND) {
73                     sonar.mode = LAND;
74                 } else if (data == WATER) {
75                     sonar.mode == WATER;
76                 }
77                 RGB(BLUE);
78                 Sonar_trigger(&sonar);
79             }
80         }
81     }
82 }
83
84 //By the point this is called the distance calculations should
85 //have occurred.
86 void I2C_TX_Callback(void) {
87     //sending it back as the u16 value. Should really do this
88     //in the library but lets just check it works first.
89     u8 high = sonar.distance.as_u16 >> 8;
90     u8 low = sonar.distance.as_u16;
91     u8 dataToSend [2] = { low, high };
92     TwoWire_writeBytes(&twi0, dataToSend, 2);
93     RGB(GREEN);
94 }

```

1.8.14 Main Motor MainMotor.h

```

1      #define F_CPU 3333333UL
2
3      #include <avr/io.h>
4      #include <util/delay.h>
5      #include <avr/interrupt.h>
6      #include "Common.h"
7      #include "StepperMotor.h"
8      #include "Wire.h"
9      #include "ShortTypes.h"
10
11     #define ADDR 0x45
12
13     void setup(void);
14     //TWI Library requires callbacks.
15     void I2C_RX_Callback(u8);
16
17     typedef enum {
18         FORWARD = 1,
19         STOP = 2,
20         BACKWARD = 3
21     } Directions;
22
23     StepperMotor stepper;
24     Directions dir;
25     TwoWire twi0;
26
27     int main() {
28         setup();
29
30         RGB(RED);
31
32         TwoWire_init(&twi0, &TWI0);
33         TwoWire_Slave_begin(&twi0, ADDR, 0, 0);
34
35         TwoWire_onReceive(&twi0, I2C_RX_Callback);
36
37         RGB(GREEN);
38         dir = STOP;
39
40         sei();
41
42         while(1) {
43             if (dir == FORWARD) {
44                 RGB(GREEN);
45                 StepperMotor_step(&stepper);
46                 StepperMotor_increaseStep(&stepper);
47             } else if (dir == BACKWARD) {
48                 RGB(BLUE);
49                 StepperMotor_step(&stepper);

```

```

50         StepperMotor_decreaseStep(&stepper);
51     } else {
52         RGB(RED);
53         StepperMotor_allStop();
54     }
55     _delay_us(750);
56 }
57
58     return 0;
59 }
60
61 void setup(void) {
62     setupRGB();
63     StepperMotor_init(&stepper);
64 }
65
66 //TWI Library requires callbacks.
67 void I2C_RX_Callback(u8 numOfBytes) {
68     if (numOfBytes == 1) {
69         while (TwoWire_available(&twi0)) {
70             u8 data = TwoWire_read(&twi0);
71             if (data == FORWARD) {
72                 dir = FORWARD;
73             }
74             if (data == BACKWARD) {
75                 dir = BACKWARD;
76             }
77             if (data == STOP) {
78                 dir = STOP;
79             }
80         }
81     }
82 }

```

1.8.15 Side Motor SideMotor.h

```

1     #define F_CPU 3333333UL
2
3     #include <avr/io.h>
4     #include <avr/interrupt.h>
5     #include "CWire.h"
6     #include "Common.h"
7     #include "ShortTypes.h"
8     #include <util/delay.h>
9
10    #define ADDR 0x46
11

```

```

12     void setup(void);
13     void setupPins(void);
14     void startMotors(void);
15     void stopMotors(void);
16     void I2C_RX_Callback(u8);
17
18     #define MOTOR1_PIN1 PIN0_bm
19     #define MOTOR1_PIN2 PIN1_bm
20     #define MOTOR2_PIN1 PIN2_bm
21     #define MOTOR2_PIN2 PIN3_bm
22
23     enum MotorState {
24         ON = 1,
25         OFF = 2
26     };
27
28     TwoWire twi0;
29     int main(void) {
30         setup();
31
32         TwoWire_init(&twi0, &TWI0);
33         TwoWire_Slave_begin(&twi0, ADDR, 0, 0);
34
35         TwoWire_onReceive(&twi0, I2C_RX_Callback);
36
37         sei();
38
39         RGB(RED);
40
41         while(1) {
42
43         }
44
45         return 0;
46     }
47
48     void setup(void) {
49         setupRGB();
50         setupPins();
51     }
52
53     void setupPins(void) {
54         PORTC.DIR |= MOTOR1_PIN1 | MOTOR1_PIN2 | MOTOR2_PIN1 |
                    MOTOR2_PIN2;
55         PORTC.OUTCLR = MOTOR1_PIN1 | MOTOR1_PIN2 | MOTOR2_PIN1 |
                    MOTOR2_PIN2;
56     }
57
58     void startMotors(void) {

```

```

59         PORTC.OUTSET = MOTOR1_PIN1 | MOTOR2_PIN1;
60         PORTC.OUTCLR = MOTOR1_PIN2 | MOTOR2_PIN2;
61     }
62
63     void stopMotors(void) {
64         PORTC.OUTCLR = MOTOR1_PIN1 | MOTOR1_PIN2 | MOTOR2_PIN1 |
        MOTOR2_PIN2;
65     }
66
67     void I2C_RX_Callback(u8 numBytes) {
68         if (numBytes == 1) {
69             while (TwoWire_available(&twi0)) {
70                 u8 data = TwoWire_read(&twi0);
71                 if (data == ON) {
72                     RGB(GREEN);
73                     startMotors();
74                 } else if (data == OFF) {
75                     RGB(RED);
76                     stopMotors();
77                 }
78             }
79         }
80     }

```
