Designing the Embedded System of an Autonomous Sailboat

Daisy Zhang
dz298@cornell.edu
Guided By - Prof. Andy Ruina
Sibley School of Mechanical and Aerospace Engineering
Cornell University

May 16, 2021

Contents

1 Introduction

The goal of our research is to develop a low-cost autonomous sailboat as a new candidate for autonomous monitoring the oceans. Such a sailboat is propelled by wind and powered by solar energy. We minimize the boat's size to reduce its cost. Because the area of solar panels on board scales with the boat size, the reduced boat size limits the available solar power. In addition, if we want the boat to travel in northern Atlantic Ocean, the problem of insufficient power will exacerbate in overcast days. A boat with limited power supply is unable to power sensors and a microprocessor while steering the sail and rudder constantly. A potential solution to save energy is that the boat will only adjust its sail and rudder servos intermittently. For the most of the time, it will charge its battery without steering.
We plan to start with building a boat that is able to autonomously sail on the Cayuga Lake. In this paper, we will discuss the design and implementation of the embedded system of the boat. We first introduce the microcontroller used on our boat, then we explore sensors, actuators, data logging devices, and ways of communications on the boat.

2 Sailboat System Design

2.1 System Architecture

Figure 1 shows the system architecture of our sailboat's embedded system. The system design contains one Atmel SAM-E70 board. We divided the whole embedded system into five subsystems: sensor system, data logging system, power system, communication system, and actuator system. Each subsystem contains related electronic devices and corresponding peripherals on the board. The details of each system are explained in the later sections.

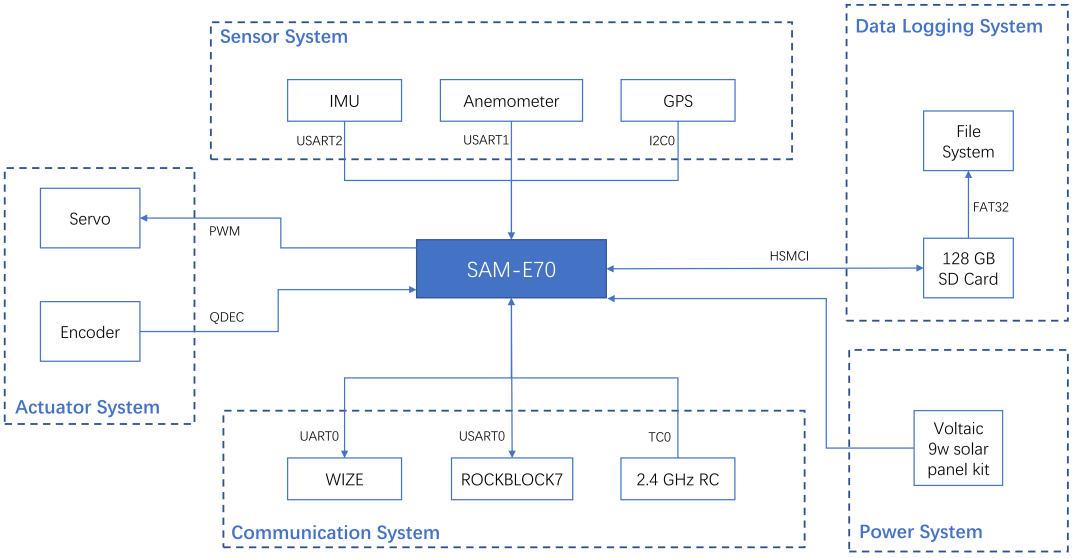

Figure 1: Sailboat Embedded System Architecture

2.2 Power Consumption

The ultimate goal of our project is to design a boat that is able to sail autonomously for months. It is necessary that our system does not run out of power. Therefore we need to be clear about the power consumption of our system. Table 1 displays the energy consumption of every electronic component on board and the board itself. More details about the maximum power consumption can be found in the data sheet of each device. For ROCKBLOCK7 and WIZE, the power consumption is different for different modes.

Component	Voltage [V]	Current [mA]	Power [W]
SAM-E70	5	90	0.45
GPS	3	23	0.069
IMU	5	40	0.2
Anemometer	9-36	$\frac{\text{Power}}{\text{Voltage}}$	0.35
WIZE	3	31(RX/Idle) 403(TX)	0.093(RX/Idle) 1.209(TX)
ROCKBLOCK7	5	40(Idle) 50(Active) 0.1(Sleep)	0.2(Idle) 0.25(Active) 0.0005(Sleep)
2.4 GHz RC	5	20	0.1
Main Sail Servo	5	150	0.75
Rudder Servo	5	200	1
Main Sail Encoder	5	16	0.08
Rudder Encoder	5	6	0.03

Table 1: Power consumption of all electronic components

From Table 1 we can calculate the total required power for our system. Before our calculation, we should make a few assumptions. We can ignore the power consumed by main sail servo and rudder servo because the boat only adjust these two servos intermittently. The time that these two servos remain on is relatively little compared to the boat's total travel time. Also we can assume that WIZE and ROCKBLOCK7 are in idle states because the communication between these devices and the base computer is intermittent as well. Given our assumption we can know the total required power is 1.572w. If we assume the embedded system of our boat is on continuously for 12 hours per day, the daily consumed energy of our boat is 18.867wh.

3 Microcontroller of the Sailboat

We use the Atmel SAM-E70 Xplained (SAM-E70 X-PLD) evaluation kit as the microcontroller (MCU) for our autonomous sailboat.

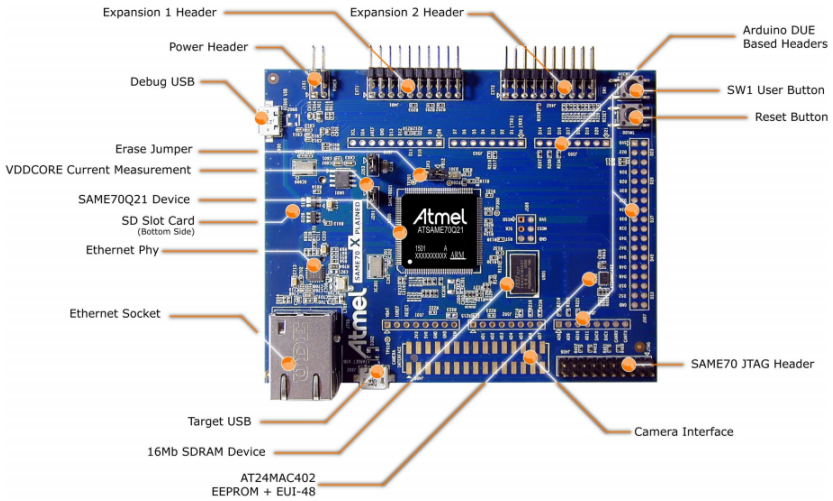

Figure 2: Atmel SAM-E70 Xplained overview from [10]

The SAM-E70 is programmed through the on-board Embedded Debugger. After the debug USB on the board gets connected to the computer, we shall launch an intergrated development environment (IDE) that supports SAM-E70 programming to start programming this board.
We use MPLAB X as the IDE for our project. Microchip Developer Help provides numerous tutorials on MPLAB X. Most of tutorials include example projects. We are able to access to the provided source files and

step-by-step instructions in completing these projects. These tutorials include but not limit to creating a project, configuring pins and building code in MPLAB X. We can learn fundamental functions that MPLAB X provides and its basic operations.
We also use MPLAB Harmony 3 in our project which can be downloaded by the framework downloader tool from the MPLAB X IDE. MPLAB Harmony is a modular software development framework for 32-bit microcontrollers (MCUs) and microprocessors (MPUs). We choose MPLAB Harmony 3 over other older releases because it adds support for SAM MCUs and MPUs where our SAM-E70 belong to. MPLAB Harmony 3 includes the MPLAB Harmony Configurator (MHC) tool. It is a set of modular, device, and middleware libraries, and has numerous example applications. MHC uses an easy-to-use Graphical User Interface (GUI) for selection, configuration, and generation of starter code, peripheral libraries, and extensive middleware (USB, TCP/IP, graphics, etc). All of them help us quickly and easily develop powerful and efficient embedded program for our board. In addition, Harmony 3 peripheral library application examples for SAM E70/S70/V70/V71 family, as the name suggests, provides examples of Harmony peripheral library applicable for our MCU. We can learn the usage of Harmony peripheral libraries from these examples. MPLAB Harmony Core Help provides help on how to use other Harmony libraries such as time system service and file system service.
To learn programming this board, we can start with a really simple program such as blinking its on-board LED. We need to read the data sheet to find the pin corresponding to the on-board LED, set the function of the pin as output, and write code to toggle this pin to achieve LED blinking. This simple program requires skills such as reading the data sheet, setting functionalities of any pin, writing and compiling code in the IDE, and uploading it to the board. We can learn these skills by following tutorials in Microchip Developer Help. Eventually we can exercise programming all vital pins on the board and we will have some reliable code that we are certain that they will work. They should be the code base for our later programming to increase our efficiency.

4 Sensors for Collecting Data

The sailboat have various sensors on board in order to collect data for monitoring the ocean/lake. There are an anemometer, a GPS, and an Inertial Measurement Unit (IMU). We use USART or I^2C peripherals for the serial communication between sensors and our board. Before interfacing each sensor to the microcontroller of the sailboat, we first connect each sensor to the computer via a TTL/USB converter. We run a few basic experiments on the sensor to test the functionality of the sensor itself, such as checking if the sensor outputs the desired data at correct baud rate. Then we connect the sensor to our MCU. We call for basic read and write function on the peripheral where the sensor connects. This call is to make sure that the wiring and the configuration of pins and baud rate are correct. After this validation, we can then write more complex code to let the sensor perform more tasks.

4.1 Anemometer

4.1.1 Anemometer Overview

The anemometer on board is TriSonica Mini from Anemoment. It is a three dimensional ultrasonic wind sensor. The anemometer not only measures wind velocity but also measures other information such as temperature, humidity, air pressure, magnetic directions, etc. The wide range of weather data measured by the anemometer makes it an excellent fit to our on-board sensors for lake/ocean monitoring. Besides, this anemometer only weighs 50 grams. This light weight can let us mount the anemometer on the top of the mast of our boat without affecting much the boat's mass distribution. This benefit furthers proves the anemometer's fitness to our project.
Figure 4 shows a full list of data that TriSonica Mini can measure, with details shown in the 'Description' column. It is worth noting that TriSonica Mini has an internal sample frequency of 40 Hz while the default output frequency is 5 Hz. The output data are averaged over the internally sampled data. Therefore, a few measure errors in instantaneous data sampling will not impact the output data much, which we will ultimately analyze.
An example of the raw output of TriSonica Mini is:

Figure 3: TriSonica Mini

Name	Description	Tagged	Tag	Decimals	Enabled	Units
IDTag	ID Tag	Yes				
S	Wind Speed 3D	Yes	S	2	Yes	m/s
S2D	Wind Speed 2D	Yes	S2	2		m/s
D	Horiz Wind Direction	Yes	D	0	Yes	Degrees
DV	Vert Wind Direction	Yes	DV	0		Degrees
U	U Vector	Yes	U	2	Yes	m/s
V	V Vector	Yes	V	2	Yes	m/s
W	W Vector	Yes	W	2	Yes	m/s
T	Temperature	Yes	T	2	Yes	C
Cs	Speed of Sound	Yes	C	2		m/s
RHTemp	RH Temp Sensor	Yes	RHST	2		C
RH	RH Humidity Sensor	Yes	RHSH	2		%
H	Humidity	Yes	H	2	Yes	%
DP	DewPoint	Yes	DP	2		C
PTemp	Pressure Temp Sensor	Yes	PST	2		C
P	Pressure Sensor	Yes	P		Yes	hPa
Density	Air Density	Yes	AD			kg/m^3
LevelX	Level X	Yes	AX			
LevelY	Level Y	Yes	AY			
LevelZ	Level Z	Yes	AZ			
Pitch	Pitch	Yes	PI	1	Yes	Degrees
Roll	Roll	Yes	RO	1	Yes	Degrees
CTemp	Compass Temp	Yes	MT	1		C
MagX	Compass X	Yes	MX			
MagY	Compass Y	Yes	MY			
MagZ	Compass Z	Yes	MZ			
Heading	Compass Heading	Yes	MD	0	Yes	Degrees
TrueHead	True Heading		TD	0		Degrees

Figure 4: A full list of data that TriSonica Mini can measure.

S 00.39 D 254 U 00.34 V 00.10 W −00.17 T 28.71 H 57.46 P 986.67 PI 000.7 RO −001.1 MD 255

Each measured information consists of a tag representing what is measured, two spaces or one space if the measured value is negative and so has a negative sign, and the value itself. What the tag represents can be found in the 'Tag' column of Figure 4. For instance, `S 00.39` means that current measured wind speed is 00.39 m/s.
Every measured datum is separated from each other by one space. The whole output message is terminated by a character `'\r'`. The output format makes parsing the output string much easier.
TriSonica Mini provides a list of commands that we can use to change the default setting of the anemometer. In particular, we are able to calibrate the anemometer according to the current temperature and the local magnetic field, to change the output frequency, internal sampling frequency, baud rate, etc. More information about commands or about the sensor itself can be found in its user manual [11].

4.1.2 Interfacing the Anemometer with MCU

This anemometer connects to our MCU through the USART1 peripheral. The configuration of this peripheral on the MCU is shown on Figure 5. The configuration of the serial port on the MCU and the wiring between the peripheral device and the MCU have to be set up correctly together in order to make our peripheral device work. The Trisonica-Mini is configured to there serial parameters: baud rate as 115200, data bits as

8, parity as none and stop bits as 1. For valid serial communication, these settings have to be matched with serial parameters of the peripheral which the anemometer connects.

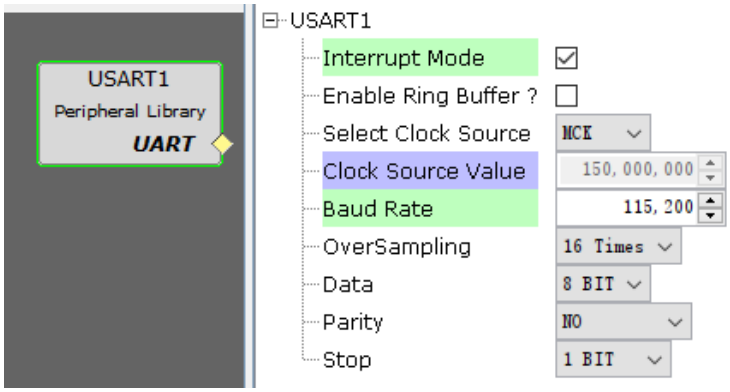

Figure 5: USART1 peripheral configuration

As for wiring, TriSonica Mini provides four un-terminated wires for user hookup. The yellow wire is 9V to 36V power; the red wire is RS-232 TxD; the green wire is RS-232 RxD and the black wire is ground.
We cannot directly connect the anemometer to our board because the anemometer complies with the RS-232 serial telecommunication standard but our MCU SAM-E70 uses TTL serial. RS-232 signals swing from -13 to +13 volts while all TTL signal levels are from 0 to V_{cc} which is 3.3 or 5 volts. By RS-232 standard, a logic low is represented by a positive voltage of a signal and a logic high is represented by a negative voltage of a signal. The rule is on the contrary for TTL. Therefore, the anemometer cannot directly interface with our MCU or it will destroy the MCU's pin. The signals from the anemometer and the board have to be inverted and regulated to each other's signal range so that the two devices can talk to each other. The solution to the problem is to plug a TTL-RS232 mutual conversion module in between the two devices. This module converts a TTL logic 0 input to between +3 and +15 V, and TTL logic 1 input to between 3 and 15 V, and vice versa for converting from RS-232 to TTL. In our project we chose the NOYITO TTL to RS232 Module.

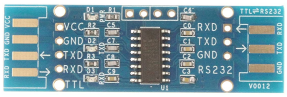

Figure 6: NOYITO TTL RS232 mutual conversion module used on our boat

This module needs to be powered while being used. The TxD on the anemometer connects to the RxD pin on the RS232 side of this module and the RxD pin of the USART port for anemometer on the MCU connects to the TxD pin on the TTL side of this module. Then the TxD of the same USART port on the MCU connects to the RxD pin on the TTL side and the RxD on the anemometer connects to the TxD pin on the RS232 side. Through this module, we can ensure valid communication between the two devices. Then we can start using basic read and write functions to read messages from the anemometer or write commands to the anemometer to change its setting. Or we can implement more complex code to complete complicated tasks such as parsing the output string from the anemometer.

4.2 Global Positioning System

4.2.1 GPS Overview

We use GNSS-4 click from MIKROE as our GPS on board [3]. The GNSS-4 click carries a SAM-M8Q module from u-blox with a built-in patch antenna. The GPS follows TTL serial so it can directly connect to the MCU without any voltage regulator.

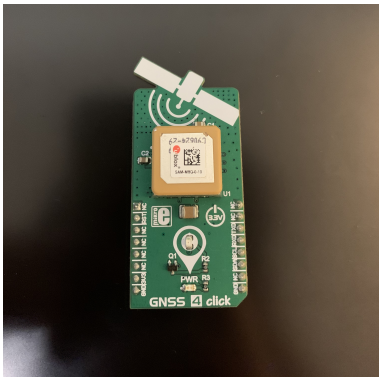

Figure 7: GNSS-4 click

The SAM-M8Q chip uses the National Marine Electronics Association (NMEA) protocol. With power on and a clear view of the sky, the GPS module outputs a packet of NMEA message at the frequency of 1 Hz. An example of a complete output packet of the GPS is:

$GNRMC,203240.00,A,4226.44379,N,07628.53814,W,0.025,,190820,,,D*74
$GNVTG,,T,,M,0.027,N,0.049,K,D*30
$GNGGA,203239.00,4226.44380,N,07628.53813,W,2,11,0.95,285.4,M,−34.5,M,,0000*7C
$GNGSA,A,3,13,15,29,18,20,,,,,,,,1.64,0.95,1.34*13
$GNGSA,A,3,81,82,75,66,67,88,,,,,,,1.64,0.95,1.34*15
$GPGSV,3,1,10,02,02,137,,05,24,079,22,13,53,058,39,15,83,136,38*71
$GPGSV,3,2,10,18,63,307,31,20,34,284,32,23,,,31,27,02,333,*4E
$GPGSV,3,3,10,29,30,209,29,51,32,221,31*76
$GLGSV,3,1,10,65,10,052,26,66,66,004,33,67,43,259,33,74,04,009,*65
$GLGSV,3,2,10,75,13,061,34,76,04,106,21,81,65,200,29,82,50,317,28*67
$GLGSV,3,3,10,83,05,334,,88,17,165,26*6E
$GNGLL,4226.44380,N,07628.53813,W,203239.00,A,D*67

This packet of messages means that the module is at position 42°26'26.63"N, 76°28'32.29"W at an altitude of 285.4 meters, moving at 0.025 knots at 2020-08-19 20:32:40 UTC.
Each line of messages begin with `$` and end with characters `'\r\n'`. The string in capital between `$` and the first comma `,` indicates this line of string belongs to which type of NMEA messages. For example, the first line of message contains `$GNRMC` so it is a RMC message which can gives us information on the GPS's position, velocity, and time when the GPS outputs this message. The spaces between commas contain different data fields. The numbers filling in these spaces are the measured numerical values of data. If there's no number between two commas, then this data field contains null information.
The GNSS-4 click outputs RMC, VTG, GGA, GSA, GSV and GLL messages. When we have a tight power budget so that we have to limit the communications between the boat and our base computer, we don't send the whole packet of GPS output messages back. We need to choose among all messages and send back the ones containing the most important information for us. RMC message could be a good candidate in this case. More information on how to decipher NMEA messages can be found in the over view of all NMEA messages[5]. This overview shows what type of information each NMEA message contains, and also shows what data fields in each message represents. We can also use any online NMEA decoder to facilitate our parsing when we have a NMEA message from our GPS, such as Free NMEA Decoder.

The u-center software from u-blox is also a power tool for performance analysis and evaluation for our GPS module [12]. This software supports SAM-M8Q that our GPS carries since it is a u-blox GNSS receiver. After the GPS connects to the computer by a UART-USB adapter, we can launch u-center and let it establish a valid connection with the GPS. Then the software is able to parse NMEA messages that our GPS outputs and visualize the information that messages contain. For example, it can plot the position reported by the GPS on a satellite map, or it can list all satellites in the GPS's view per second. The software makes evaluation for the GPS module easier. Figure 8 shows the trajectory of the GPS when I stood still for ten minutes in my backyard while carrying my computer and the GPS. From the figure, we can see that the GPS's measured positions moves within a 5-meter diameter circle. Therefore we can claim that with a good sky view (since it was measured in the backyard) and with the GPS still, the accuracy of the GPS is within 5 meters. It is within the accuracy range listed on the data sheet of SAM-M8Q, which is 2.5 to 8 meters [9].

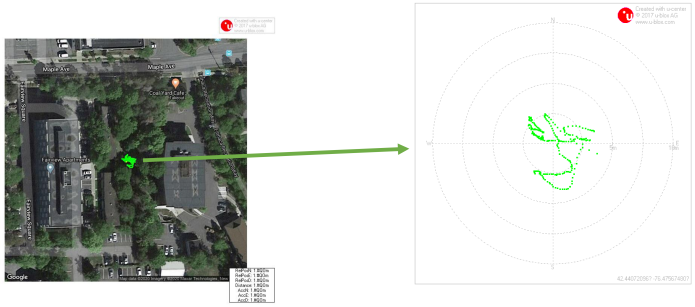

Figure 8: GPS

4.2.2 Interfacing the GPS with MCU

This GPS module connects to our MCU through the I^2C peripheral. The configuration of this peripheral on the MCU is shown on Figure 9. Figure 10 is the pinout diagram of our GPS which provides information on wiring. We need to power the GPS module and connect I^2C clock pin (SCL) and I^2C data pin (SDA) to corresponding pins on our MCU. The corresponding I^2C pins on the MCU can be found and enabled by the pin configuration tool of MPLAB Harmony.
It is noteworthy that when we call read or write functions on the I^2C peripheral, these functions require the address of the SAM-M8Q module on the GNSS-4 click. The default address of SAM-M8Q and all u-blox GPS products is at 0x42.

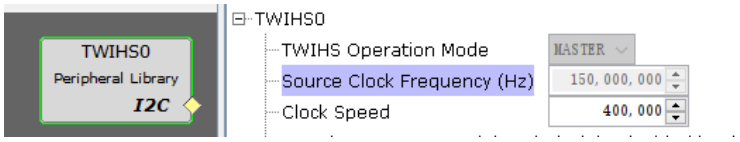

Figure 9: I^2C peripheral configuration

4.3 Inertial Measurement Unit

4.3.1 IMU Overview

The Inertial Measurement Unit (IMU) on board is a VN-100 SMD Development Kit (Thermal Calibration) from VectorNav. The development kit carries a thermally calibrated VN-100 on board. The thermal calibrated feature enables the IMU to meet performance specifications over the temperature range of -40 C to +85 C [15]. The operating temperature range can better satisfy the marine working environment of our sailboat.

Notes	Pin	mikroBUS				Pin	Notes
	NC	1	AN	PWM	16	NC	
Active Low	**RST_N**	2	RST	INT	15	NC	
	NC	3	CS	TX	14	**TXD**	UART transmit
	NC	4	SCK	RX	13	**RXD**	UART receive
	NC	5	MISO	SCL	12	**SCL**	I2C Clock
	NC	6	MOSI	SDA	11	**SDA**	I2C Data
Power supply	**+3.3V**	7	3.3V	5V	10	NC	
Ground	**GND**	8	GND	GND	9	**GND**	Ground

Figure 10: GPS pinout diagram

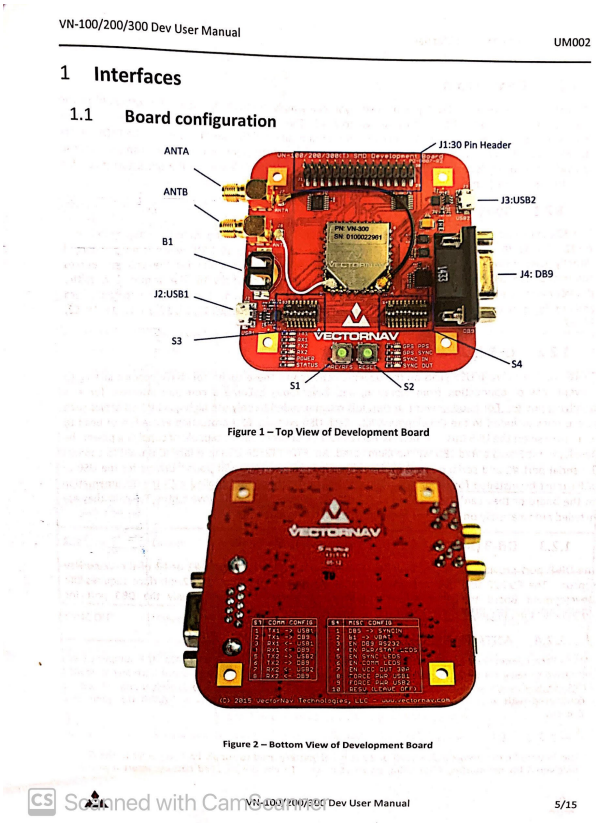

Figure 11: IMU board configuration, scanned from VN-100 Dev User Manual

To get started on this IMU, we can connect a USB cable to J2:USB1 and to our laptop. After a new COM port becomes available, we can use a terminal program such as Tera Term to monitor the sensor's output. The configuration of this COM port should be baud rate as 115200, data bits as 8, parity as none and stop bits as 1.
An example of the raw output of the IMU is:

$\$VNYMR, -053.157, +000.413, -002.022, +00.0794, +00.1220, +00.4311, +00.065, +00.327, -09.364, -00.001074, +00.000689, -00.000518*61$

Where `VNYMR` is the output header. It indicates that this message has yaw, pitch, roll, magnetic, acceleration, and angular rate measurements in x, y, and z directions of the IMU. The orientation of x, y and z-axis are carved on the lower left corner of the VN-100 on the center of the development kit. The spaces between two

commas are different data fields and contain the measured values of the data. All output messages start with a dollar sign, followed by a five character command, a comma, command specific parameters, an asterisk, a checksum, and a newline character. There are other headers such as `VNACC` and `VNYMR`. Each of the output header indicates that the output message contains a different set of measurements. The full list of all headers along with their corresponding set of measurements can be found in the user manual at page 60 [15].
In addition to terminal programs, we can also use VectorNav Control Center to explore our IMU. It is a powerful software for the VectorNav IMU. It provides an assortment of tools for us to visualize and inspect the measurements streaming from the IMU. This tool also can let us to easily set up the IMU, change its settings or configure its pins. It can also serve as a data logger and exporter. The Control Center is able to log all data streaming from the sensor and communications between the computer and the sensor in a single file for later analysis. Through the Control Center, any measurements from the sensor can be exported to a CSV file, a text file, or a MATLAB workspace.
VectorNav provides programming libraries in different languages for VN-100. The programming libraries can be downloaded from VectorNav Programming Library. Languages include C, C++, Labview, Matlab, .Net, Python and Unity [13]. Each library contains detailed documentation and example project. We can easily interface the sensor with our board by making use of functions from the programming libraries.
For further information in product design and development on this IMU, the following manuals can be consulted and can be obtained by contacting VectorNav: VN-100 User Manual [15] and VN-100 Dev User Manual [14]. The user manual of the development kit and the user manual of VN-100 are two separate manuals. They together provide the IMU's physical and electronic specifications, all types of output data that it measures and outputs, and all commands that it accepts.

4.3.2 Interfacing the IMU with MCU

This IMU connects to our MCU through the USART2 peripheral. The configuration of this peripheral on the MCU is shown on Figure 12: baud rate as 115200, data bits as 8, parity as none and stop bits as 1.

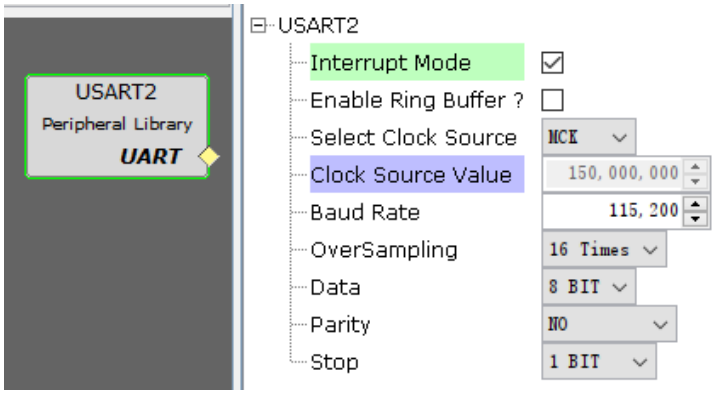

Figure 12: USART2 peripheral configuration

The IMU uses serial TTL as its digital interface so it can directly connect to the SAM-E70 board. The IMU's pin descriptions can be found on page 12 in VN-100 Dev User Manual [14]. We need to power the IMU and connect Tx and Rx pins on the IMU to corresponding USART2 Rx and Tx pins on our MCU.

5 Data Logging

5.1 SD Card Logging

We want to have an active data logger on board during our boat's mission. This logger should record data measured by sensors over time, the communication messages between the base computer and the boat, and any possible error messages. We also want the data logger to have enough space to store any pictures or videos taken by a on-board camera which we may mount to our boat in the future.

Therefore we choose a SanDisk SD card for data logging while its storage space is 128 GB. The SAM-E70 X-PLD has a standard SD card slot. This slot is a SD/MMC J600 Socket and the pins associated with the slot connects to the High-speed Multimedia Card Interface (HSMCI) peripheral of the board. We use MHC to set up these pins and a file system so that we can perform file operations on the SD card (open a file, read the file, write to the file, etc).

Figure 13: SD card in the card slot of SAM-E70

We use Project Graph and Pin Configuration these two tools from MHC to set up the file system of the SD card. We put the SDMMC driver to Project Graph from Driver subcategory of Harmony category in available components. Then we pick HSMCI from Peripherals category and FILE SYSTEM from System Service subcategory of Harmony category to Project Graph. Then we connect them as shown in Figure 14.

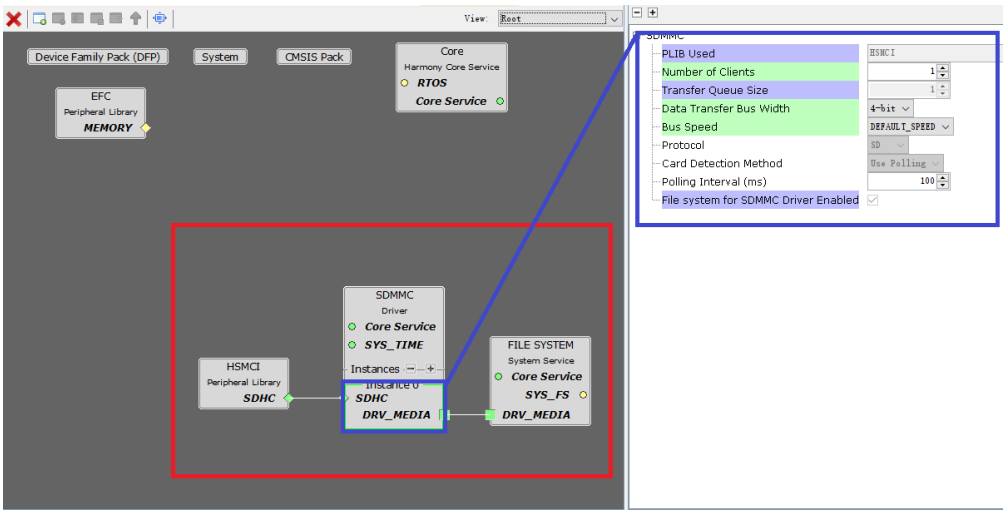

Figure 14: The components in red rectangle show the MHC configuration for connection between the SDMMC driver, HSMCI peripheral and FILE SYSTEM system service. The part in blue rectangle is the configuration option for the instance 0 of the SDMMC driver. Its instance 0 of connects the HSMCI interface on board to the File System service. The configurations of HSMCI and FILE SYSTEM remain default so they are not shown in this screenshot.

Then we use Pin Table in Pin Configuration to enable all pins in HSMCI module so that all pins with functions for HSMCI clock and signal transmitting are enabled. In addition, we also have to enable the card detect pin on board (PC16). If it is not enabled, our board cannot detect whether the SD card is inserted

so any file operations would result into error. All the pins mentioned above along with their function descriptions can be found in the user manual of SAM-E70 at page 34 [10].
There is one more essential step but we don't use MHC here. The file system service of our board support multiple file systems include FAT12, FAT16, FAT32 and the Microchip File System (MPFS) among others. MPFS is read-only so we don't consider it for our data logger. Therefore we only can use FAT12, FAT16 or FAT32 as our file system to access the SD card through our board. However, our SD card has a storage space of 128 GB. Any SD cards larger than 32 GB automatically adopts exFAT file system which is not supported on our board. The inconsistency between the file system adopted by our card and the file system service supported by our board is a problem that we cannot solve with MHC.
Therefore we need partition our SD card into volumes so that each of the volume is equal or less than 32 GB. We can insert our card to a SD card reader and connect the reader with the computer. Then we can partition the card drive with Disk Management if we are on Windows. There are many how-to-partition-your-drive tutorials online so we don't outline the whole process here. We divide our card into four partitions while each partition has a space of approximate 32 GB. They no longer adopts exFAT file system but automatically adopts FAT32 because of the size change.
After the setup, we generate code by MHC based on our configuration made in Project Graph. MHC generates several configuration files and also programming libraries or APIs for us to access to the SD card and do file operations. A successful file operations require multiple steps. We need to first mount the card, then set current drive, open the desired file, read or write to the file, and finally close the file. Each step has an associated function in the programming library for file system that we generate. The documentation for all the functions can be found in MPLAB Harmony Core Help under File System Service Library Help. The state machine in Figure 15 fully describe how to access to the card and perform file operations. Note that our SD card has four partitions. If a SD card with four partitions is attached to the system, and assuming all four partitions are recognized, there will be four device to be mounted: mmcblka1, mmcblka2, mmcblka3 and mmcblka4. Therefore we need to mount all the devices and name their drives differently with `fmount` function. When we have different partitions, we also have to make sure if the current drive is set correctly before visiting files.

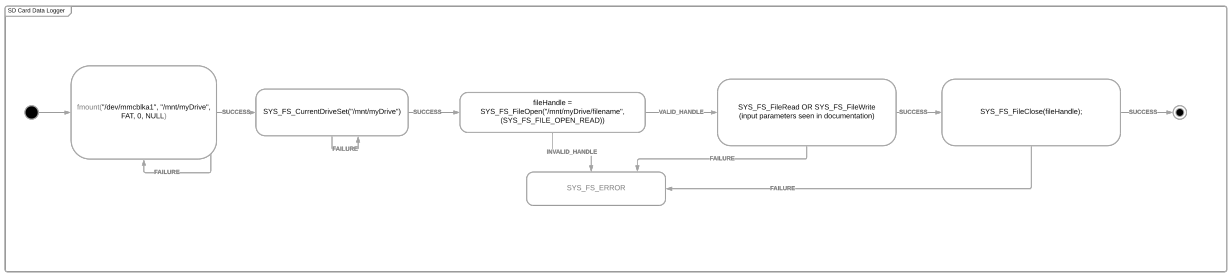

Figure 15: The definite state machine for access to SD Card.

6 Communication System

We have three ways of communication between our autonomous boat and the base computer. They are radio frequency (RF) communication, satellite communication and remote control (RC). The RF communication and the satellite communication are mainly for transceiving binary or ASCII messages. The remote control is a backup solution to control the boat to maneuver back ashore.

6.1 Radio Frequency Communication

6.1.1 RF Module Overview

We use a pair of Radiocrafts development boards carrying RC1701HP-WIZE for our RF communication. The RC1701HP-WIZE is a Wize compliant RF module. The module contains a communication controller

with embedded Wize protocol as specified by the Wize Alliance based on Wireless M-Bus operating at 169 MHz.
The development boards were purchased as a development kit (DK). The kit is designed for fast prototyping and easy evaluation of the onboard module. The kit includes two development boards, antennas, USB cables and power adapters. The development board contains RC1701HP-WIZE module with associated support circuits and digital IO pins. The board can easily be operated using a PC terminal emulator or the RCTools PC suite from Radiocrafts. A quick start guide on setting up a link between our two development boards and testing their functionality are as follows [6].

- Attach the antenna to the SMA connector onboard
- Download the latest version of Wireless M-Bus RCTools from Radiocrafts
- Connect the USB port to a PC which provides power and a serial port
- Start MBUS_CCT tool from RCTools
- Select the correct COM-port and set data rate 19200, 1 start bit, no parity, 1 stop bit, no flow control
- Repeat the steps above for the other development board

Now the two modules will be both in idle mode ready to transceive a valid data packet. We can enter data in one terminal window and wait for a timeout. If the transmission is successful, we will see the data shown in the other terminal window. This indicates that the data is transmitted to the other module.
However, there are formats for transmitting and receiving data. Using these formats correctly are important in transceiving a valid data packet. We will discuss this in subsubsection 6.1.3.
In addition to the idle mode, the module also has config mode, and sleep mode. When the module is in config mode, its configuration can be changed by sending a set of commands on the UART interface. Sleep mode can save power. Details on entry to these modes and the overview of Wize configuration commands can be found in the WIZE user manual [16].

6.1.2 Interfacing RF module with our MCU

We connect one of the development boards with our base computer by connecting its USB port to the PC. Then we connect the other development board with our SAM-E70. We no longer use USB to power this board but there's a 9V battery connector on the bottom side available for powering. The RXD and TXD pins on the development board go to the corresponding TXD and RXD pins on our SAM-E70. Meanwhile, the ground pin on the development board has to be grounded as well. All pins on the board are accessible in standard pin rows with a spacing of 2.54 mm. Detailed pinout of our development board could be found in the DK user manual [6].

6.1.3 WIZE Message Format

The data frame for transmitting a Wize packet is built like Figure 16.

Figure 16: Frame format for transmitting data. This image is from [16]

L is the first byte sent to the module and it is the length byte. It is the length of the data packet without including the length byte itself. The maximum length of the message transmitted to the module is 0xF6 while the minimum length is 0x01. The length byte values larger than 0xF6 have been given special meanings and are defined as soft commands because they are commands that can be sent when the module is in idle mode and not in configuration mode.
CI is the Control Information byte. The value 0x20 has been used as the CI byte to indicate that the embedded protocal is Wize.

The application data following CI is the actual data.
At the end of the message, TS is the last 2 bytes and is the timestamp value. The timekeeping must be done by the host MCU, our SAM-E70, because RC1701HP-WIZE does not contain a time counter. If a real-time clock is not available these two bytes could be set to 0x00.
The module receives the data packet over UART interface and is ready to transmit this message over RF. The link layer address (LLA) and C-field (and adjusted L value) is added to the Wize packet automatically by the module before transmitting over RF, and both can be changed in configuration mode. The module also adds the authentication tags (MAC1 and MAC2) and the timestamp (TS). Finally, the module will add the link layer CRC.
For example, if we want to transfer data `0xA0B0C0D0E0F0` without timekeeping, we need to transmit `0x0920A0B0C0D0E0F00000` on the UART interface of our RF module. The module will then automatically add LLA, C-field, and other bytes and send this modified Wize packet over RF.
The data frame for receiving a Wize packet is built like Figure 17.

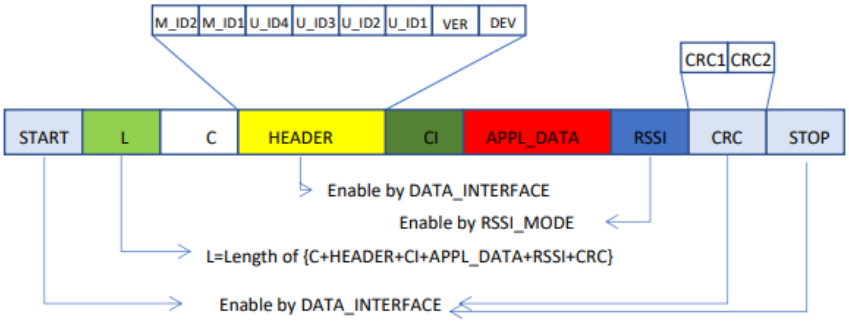

Figure 17: Frame format for receiving data. This image is from [16]

Data in light and dark blue, and yellow are optional fields and can be enabled or disabled by setting the DATA_INTERFACE and RSSI_MODE parameters in configuration mode.
L is the length byte. It is the length of the message without including itself, START and STOP bytes.
C is the second byte received by the module if START, CRC and STOP bytes are disabled. The default value is 0x44.
HEADER consists of Manufacture ID, Unique ID, Version and Device Type of the RF module. It is used on the master side to bind slaves to this master. By default, the value of HEADER is 0x2448785634120100.
CI is the control information byte, same as the CI in the data packet transmitted.
APPL_DATA contains the actual data, timestamp and other bytes added by the module transmitting the Wize packet.
RSSI is added by the module by setting RSSI_MODE = 1. It is an indication of the signal strength of the received packet.
We can use MBUS_DEMO, a software from RCTools suite to better understand valid Wize packet formats. RCTools include MBUS_CCT and MBUS_DEMO. We haved mentioned MBUS_CCT in the quick start guide where it is used as a terminal emulator. Both softwares requires access to the modules UART via an available COM-port. MBUS_DEMO provides Module Quick setup, MBUS Packet generator, MBUS Packet Sniffer and an Installation tool for the MBUS2/MBUS3 feature set.
The MBUS Packet generator allows us to set up one of our RC1701HP-WIZE development board as a transmitter that sends valid data packets periodically. The data packets are generated by the MBUS Packet Generator and are of different lengths.
The MBUS Packet Sniffer allows us to set up the other development board as a receiver that receives these data packets. The program can count the total number of received MBUS packets and list them in the log list. The log list displays values of all fields of each message received.
Therefore MBUS_DEMO can well demonstrate Wize packet formats for us. The detailed instructions on how to use MBUS_DEMO can be found in its user guide [4].

6.2 Satellite Communication

6.2.1 RockBLOCK 9603 Overview

We use the RockBLOCK 9603 carrying an Iridium 9603 modem for our satellite communication. The RockBLOCK module provides the modem with an antenna and supporting circuits. It also provides users the access to the modem's serial interface via a row of pins.
Iridium is the only satellite network that allows data transmission from any point of the world while other network doesn't cover the polar region and marine areas fully [7]. Iridium has 66 satellites around the Earth. The amount of satellites ensures a full coverage of data communication service on Earth 24/7 [7]. With the Iridium 9603 modem, our RockBLOCK module enables our boat to send and receive short messages from anywhere successfully with a clear view of sky. Messages sent take only seconds to reach their destinations. In order to use the data transmission service of Iridium, we need to purchase credits and line rental. Line rental is paid in the unit of month and we only need to pay for months when we want to use the RockBLOCK to send or receive messages. In the months when line rental are purchased, the RockBLOCK can transceive messages via the Iridum satellite network and we users can access to the RockBLOCK management system to managing our RockBLOCK device. Credits are used every time when data transmission via Iridium networks happens. 1 credit is used per 50 bytes or per message sent or received if the message is less than 50 bytes. 1 credit is also used when the RockBLOCK uses mailbox check function to check if there's any new incoming messages queued. Credits must be used in the months when line rental is purchased, and they don't expire until there's no data transmission activity associated with the RockBLOCK account for 12 months.

6.2.2 Interfacing the RockBLOCK with MCU

Figure 18 and Figure 19 show the modem side and the antenna side of the RockBLOCK 9603. It is important that when we use the RockBLOCK 9603, the antenna should be positioned looking upwards to get a clear view of sky. Otherwise, data transmission always fails. Also, we cannot install the RockBLOCK inside a solid opaque enclosing because the microwave L-band from the RockBLOCK module cannot penetrate solid objects.

Figure 18: RockBLOCK 9603 - Modem side. This image is from [8]

This RockBLOCK module connects to our MCU through the UART0 peripheral. The configuration of this peripheral on the MCU is: baud rate as 19200, data bits as 8, parity as none and stop bits as 1.

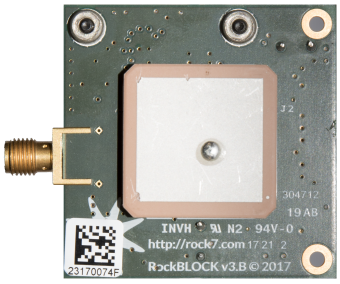

Figure 19: RockBLOCK 9603 - Antenna side. This image is from [8]

The module uses serial TTL as its digital interface so it can directly connect to the SAM-E70 board. The module's pin descriptions can be found in the developer guide [7]. We need to power the module and connect Tx and Rx pins on the module to corresponding UART0 Rx and Tx pins on our MCU.

6.2.3 The RockBLOCK Data Cycle

In this section, the data processing cycle will be outlined to show how the RockBLOCK module handles data.
We attach our RockBLOCK 9603 to the SAM-E70 board. The RockBLOCK 9603 carries an Iridium 9603 modem which is able to transceive packets of ASCII and Binary messages. It can transmit messages in packet of up to 340 bytes and receive messages in packet of up to 270 bytes. With the same amount of information, Binary messages save more spaces compared to ASCII messages and thus spend less credits. Binary message can squeeze in more information per transmission but it requires conversion between the original form of our data.
We can talk to the RockBLOCK module to transceive data using a simple set of AT commands whose function library can be found in the AT command Reference Book. Code examples are in Web Service Guide.
Messages transmitted from the RockBLOCK module, or from the boat in our project, are defined as Mobile Originated (MO). It means that these messages originate from our mobile device. Messages transmitted to the RockBLOCK module are defined as Mobile Terminated (MT) because they terminate to our mobile device.
Data transmission with the RockBLOCK module is a two-step process. First, data to be transmitted must be transferred from our controller to the MO buffer of the RockBLOCK module. Then a send/receive command, as known as an SBD session, is executed. The SBD session sends the MO message to its destination and receives the MT message waiting in line. The session also gives information about the status of the MO messages just sent and the total number of unread MT messages waiting in line.
After the SBD session, data in the MT buffer needs to be transferred from the RockBLOCK module to our controller.
Details about this process and associated code examples are outlined in the Transmit ASCII data, Transmit binary data and Receive data chapters of the Web Service Guide.
It generally takes around 20 seconds from power-up of the RockBLOCK module to a successful data transmission, with a clear view of sky. It may take up to several minutes with a restricted view [7].
Messages sent from the RockBLOCK module can be delivered to our chosed email address or to a web service as a HTTP POST. The message will be hex encoded so there are no character set problems. Full details of our web service are available in the Web Service Guide.

6.3 Remote Control

The Remote Control is designed to be a backup solution for this project. If there's a system failure, we can use the RC to control the boat to maneuver back ashore. We use a 2.4 GHz RC transmitter and a 6-channel receiver for remotely controlling our boat. The receiver receives a 2.4 GHz signal from each channel of the transmitter, converts it to a PWM signal and outputs the PWM signal in the corresponding channel. The output channels are connected to pins on our board SAM-E70. The SAM-E70 will measure the pulse widths of these signals and control our actuators based on these measurements. One of the signals acts as a switch to active or deactivate the remote control. The SAM-E70 keeps observing the pulse width of this signal. Once it exceeds a threshold, the boat activates the RC so that we can control the RC transmitter to steer our boat back. Otherwise, the boat performs its control system and sail autonomously.

Figure 20: Flysky 6-channel transmitter and receiver used in our project. This image is from [2]

We use timer counters in capture mode to measure the pulse widths of PWM signals from the RC receiver. An example of the configuration of timer counter 0 in capture mode is shown in Figure 21. In this example, TC0 will capture the time counter value at the rising edge of the signal from TC0's TIOA0 pin, and capture the time counter value at the falling edge of this signal. The difference between the two values times the timer resolution is the pulse width of the signal. Therefore, for instance, if we want to measure the pulse width of the PWM signal from Channel 6 of the RC receiver, we need to first enable TC0's TIOA0 pin onboard and then connect the signal pin on Channel 6 to TC0's TIOA0 pin. If we want to use other timer counters and different channels, say TCx and Channel y, the corresponding pin will be TCx's TIOAy pin.

7 Power System

We use a solar panel with a Li-Ion USB battery as the power supply to our boat. The solar panel has a peak power of 9 watts while the battery has a capacity of 47 watt hours. The battery outputs 5 volts constantly and can provide a current within 2 to 3 amperes.
We choose this solar panel and the battery based on the power consumption of our boat. As we have already discussed in subsection 2.2, the total required power is 1.572w. If we assume that the embedded system is on without any sleeping for 12 hours per day, the daily energy consumed is 18.867wh.
We use the PV performance tool provided by Photovoltaic Geographical Information System to estimate the performance of our solar charger kit on powering our system. This tool is able to calculate the performance of PV systems that are not connected to the electricity grid but instead rely on battery storage to supply energy when the sun is not shining.

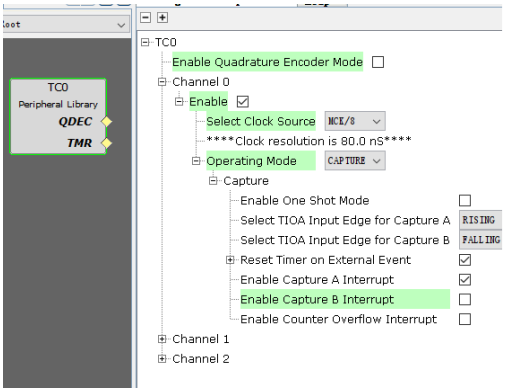

Figure 21: TC0 in capture mode configuration

Figure 22: Voltaic 9 Watt Solar Charger Kit used in our project. This image is from [1]

We input the peak power of our solar panel, the battery capacity, and daily energy consumption to this tool. And we select Cayuga Lake as the location so that the solar irradiance data at this place will be used in the performance estimation. All other inputs remain default.
The result is shown in Figure 23. The bars in dark blue labeled as energy output is the energy out of the battery and used by our system. The bars in light blue labeled as energy not captured is the energy captured by the solar panel but not stored in the battery. From the result, we can know that from April to September, our system could be on for 12 hours continuously per day without running out of power. Moreover, in real applications, some components of the embedded system will be in sleep mode occasionally to further save power. Because we plan to launch our boat on the lake in the summer, we can claim that our solar charger kit is suitable for our embedded system.

8 Future Development

So far we have discussed all subsystems of the sailboat's embedded system except the actuator system. The actuator system has not been implemented yet and should be completed in the near future to enable our boat to sail on the Cayuga Lake.
The actuator system includes the main sail servo, the rudder servo and their corresponding encoders. The servos will be controlled by PWM signals with different duty cycles emitted from pins on the SAM-E70 board. Signals from each encoder will be fed to timer counters in QDEC mode of our MCU to decode

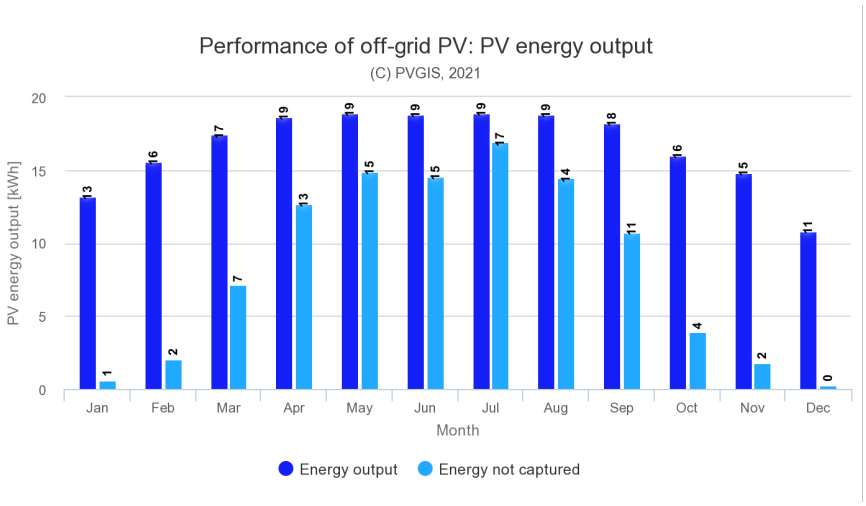

Figure 23: Performance estimation of our solar charger kit, generated by Photovoltaic Geographical Information System.

current position and moving direction of the encoder. The motor control algorithm needs to be implemented so that positions known from the encoders, weather data provided by sensors and the preset destination location will act as input and the desired sail and rudder angles and associated PWM signals to control servos act as output.
In addition, we also have to implement the scheduler of the embedded system. The scheduler should allocate each subsystem a specified amount of time to perform its function. Once the time period expires, the scheduler immediately exits the function being executed by the subsystem and starts the next subsystem. Each time when the function of a subsystem is called, it should starts from the state last time when it exited. There should be no busy waiting or forever loop in each subsystem.

9 Conclusion

We plan to develop a low-cost autonomous sailboat that act as a new candidate for ocean monitoring. Our first prototype is a boat that is able to sail autonomously on the Cayuga Lake. This paper discuss the details and implementation of the sailboat's embedded system. We have divided the embedded system into five subsystems. Each system contains electronic devices, peripherals and code. We have implemented four subsystems out of five: sensor system, data logging system, power system, and communication system. The actuator system should be implemented in the near future. We should also implement the scheduler of the embedded system so that each subsystem is allocated with a specific amount of time to execute its function per loop with no busy waiting or forever loop. Once the boat is completed, we will launch it on the Cayuga Lake and let it sail to the destination set by us and sail back. During this mission, the performance of the boat and data collected by onboard electronic components can inform us critical design decisions in the making of the future boat for ocean monitoring.

References

[1] *9 Watt Solar Panel Kit with USB Battery Pack*. URL: https://voltaicsystems.com/9-watt-kit/. accessed: 05.02.2021.

[2] *Flysky FS-i6X 6-10(Default 6)CH 2.4GHz AFHDS RC Transmitter w/ FS-iA6B Receiver*. URL: https://www.amazon.com/Flysky-FS-i6X-Transmitter-FS-iA6B-Receiver/dp/B0744DPPL8/ref=pd_bxgy_img_2/141-1222861-6444501?_encoding=UTF8&pd_rd_i=B0744DPPL8&pd_rd_r=d72442bc-0ac1-4b24-9e15-cfa2545e43d6&pd_rd_w=9V3aX&pd_rd_wg=hWgM6&pf_rd_p=fd3ebcd0-c1a2-44cf-aba2-

bbf4810b3732&pf_rd_r=TPZR0GQKZGBYDQ9GXTVW&psc=1&refRID=TPZR0GQKZGBYDQ9GXTVW. accessed: 04.27.2021.

[3] *GNSS 4 click - board with SAM M8Q module from u-blox*. URL: https://www.mikroe.com/gnss-4-click. accessed: 09.13.2020.

[4] *MBUS-DEMO User Manual*. URL: https://www.radiocrafts.com/uploads/mbus-demo_user_manual_1_30.pdf. accessed: 05.05.2021.

[5] *NMEA-0183 messages: Overview*. URL: https://www.trimble.com/OEM_ReceiverHelp/V4.44/en/NMEA-0183messages_MessageOverview.html. accessed: 09.13.2020.

[6] *RC11xxHP/RC12xxHP/RC17xxHP-RC232/MBUS-DK User Manual*. URL: https://radiocrafts.com/uploads/RC11xxHP_RC12xxHP_RC17xxHP-RC232_TM_MBUS-DK_User_Manual.pdf. accessed: 05.05.2021.

[7] *RockBLOCK 9603 Developer Documentation*. URL: https://docs.rockblock.rock7.com/docs. accessed: 05.12.2021.

[8] *RockBLOCK 9603 Developer Documentation Photographs*. URL: https://docs.rockblock.rock7.com/docs/photographs. accessed: 05.12.2021.

[9] *SAM-M8Q Datasheet*. URL: https://www.u-blox.com/en/docs/UBX-16012619. accessed: 09.13.2020.

[10] *SAME70-XPLD User Guide*. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-44050-Cortex-M7-Microcontroller-SAM-E70-XPLD-Xplained_User-guide.pdf. accessed: 09.13.2020.

[11] *TriSonica-Mini User Manual*. URL: https://anemoment.com/wp-content/uploads/2019/04/Trisonica-Mini-User-Manual-Ap2019.pdf. (accessed: 09.12.2020).

[12] *u-center — u-blox*. URL: https://www.u-blox.com/en/product/u-center. accessed: 09.13.2020.

[13] *VECTORNAV PROGRAMMING LIBRARY*. URL: https://www.vectornav.com/resources/vectornav-programming-library. (accessed: 10.17.2020).

[14] *VN-100 Dev User Manual*. URL: https://drive.google.com/file/d/1qCqBvLkh-erCyHhrIsA8-Yn1sUq83c9T/view?usp=sharing.

[15] *VN-100 User Manual*. URL: https://www.vectornav.com/docs/default-source/user-manuals/vn-100-user-manual-rev_2-45.pdf?sfvrsn=5049c8ff_4. (accessed: 10.15.2020).

[16] *WIZE User Manual*. URL: https://radiocrafts.com/uploads/WIZE_User_Manual.pdf. accessed: 05.05.2021.

A Code Cloud Storage

The code so far implemented to control our sailboat are stored in two different GitHub repository. The first one is sailboat$_m$*plabx*. It contains runnable code for controlling our boat with all the configuration files, peripheral APIs, and the main script. The second one is autononomous-ruina. This repository contains only code to control each electronic component of the embedded system without configuration files. Code is made to be modular so that it can be applied to similar projects with minor edition. Details about each repository is in the Readme file of each repository.