*Scientific contribution/Original research*

# Open Science: Development of Open Platform for Giant Unilamelar Phospholipid Vesicles Electroformation

**Gazvoda de Reggi M[1], Malavašič U[1], Jeran M[1,2], Penič S[1,*]**

1. University of Ljubljana, Faculty of Electrical Engineering Laboratory of Physics, Tržaška cesta 25, SI-1000 Ljubljana
2. University of Ljubljana, Faculty of Health Sciences, Laboratory of Clinical Biophysics, Zdravstvena pot 5, SI-1000 Ljubljana

* Correspondence: Samo Penič; samo.penic@fe.uni-lj.si

**Abstract:**
Open Science is, by definition, the transparent and accessible knowledge that is shared and developed through collaborative networks. It allows for broader availability of research data and speeds up the process of obtaining new knowledge. Open-source software has publicly available source code that anyone can inspect, modify, and enhance. This work focuses on the importance of Open Science practice and illustrates that on an example of cell membrane research. Cell membranes are vital components of the cell as their structure and properties influence many important cellular functions. Due to their complicated composition, researchers use simpler structures that resemble the membrane, called unilamellar vesicles, especially giant unilamellar vesicles, due to their size. There are many methods for acquiring the vesicles using their main building blocks – phospholipids, electroformation being the most commonly used. In this work, an open-source sinusoidal voltage source designed for electroforming giant unilamellar vesicles will be presented as an alternative to expensive and impractical function generators. The device has been successfully used to produce vesicles from the phospholipid POPC (1-palmitoyl-2-oleoyl-*sn*-glycero-3-phosphocholine) and a mixture of POPC and cholesterol (4/1).

**Keywords:** Open Science, Open Source, Arduino AC voltage source, Giant unilamellar phospholipid vesicles (GUVs), Cell membranes, Electroformation.

**Citation:** Marin Gazvoda de Reggi, Urban Malavašič, Marko Jeran, Samo Penič. Open science: development of open platform for giant unilamelar vesicles electroformation. Proceedings of Socratic Lectures. 2021; 6: 99-113. https://doi.org/10.55295/PSL.2021.D.014

## 1. Introduction

### *1.1. Open science*

Open Science is, by definition, the transparent and accessible knowledge that is shared and developed through collaborative networks. There are six fundamental principles of open science, which include open methodology, open source, open data, open access, open peer review and open educational resources. Open science practice has many advantages. It allows for broader availability of research data and it also has the potential for speeding up the process of obtaining new knowledge (Vicente-Saez and Martinez-Fuentes, 2018).

### 1.2. *Open-source hardware and software*

The term open source refers to something that is freely available for modification and redistribution because its design is publicly accessible. It most commonly represents open-source software with source code that anyone can inspect, modify, and enhance. Open-source technology not only benefits computer programmers but practically everyone –Linux kernel is one such example as its variations can be found on a wide range of devices ranging from smallest wearables, smart household appliances, personal computers to giant servers and supercomputers (Opensource.com, 2021).

In a similar manner, open-source hardware adopts the open-source principles by making design specifications of a physical object (which include schematics, blueprints, logic designs) publicly available so that they can be studied, modified, created, and distributed by anyone. Preferably its components should be easy for anyone to obtain which reduces common barriers to the design and manufacture of physical goods. Arduino and Raspberry Pi are among the most renowned examples (Opensource.com, 2021).

### *1.3. Scientific Computing*

Scientists are frequently presented with various tasks which include computation and managing large amounts of data. Instead of doing repetitive action by hand, scientists utilize the help of computers and typically develop their own software for such purposes, as it requires significant domain-specific knowledge. The resulting software is often buggy and difficult to maintain since most are self-taught. In order to improve the quality of code, the subsequent practices are to be followed: improving the human readability of code (distinctive and consistent naming, formatting), automating workflows and repetitive tasks, making incremental changes (as well as using a version control system), reusing and modularizing code instead of rewriting it, planning for mistakes and optimizing it only after it works correctly, collaborating and documenting (Wilson et al., 2014).

### *1.4. Cells and their membranes*

We chose the preparation of cell membrane substitutes in a laboratory as an example of what can be achieved by following the principles of open science and open-source software and hardware as well as best practices for scientific computing (Gazvoda de Reggi et al., 2021).

Cells are the main building blocks of all living organisms. Their exterior and interior are separated by a thin layer called the cell membrane, through which substances pass into and out of the cell. The cell membrane is primarily composed of phospholipid molecules organized into a bilayer (Cevc and Marsh, 1987). It also consists of various inclusions, such as different types of proteins, cholesterol and other biologically important building blocks that define cell's function (**Figure 1**).

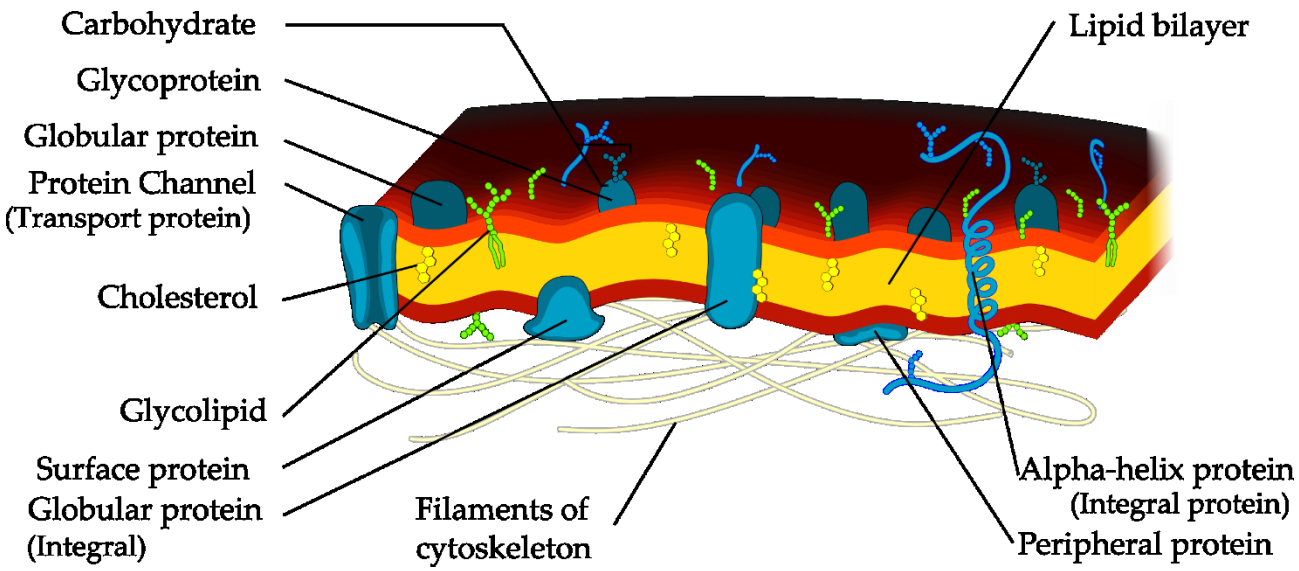



**Figure 1**. The cell membrane is a complex structure whose basic building block is a phospholipid bilayer. The two red-colored layers are composed of many polar phospholipid heads, while the yellow-colored area represents the non-polar tails of the phospholipid molecules (Wikipedia Contributors, 2021).

The most important functions of a membrane are to protect the inside of the cell from external influences and to regulate the transport of substances in and out of the cell. The structure of the membrane thus determines the specific functions of the cell. In the interior of the cell, we also find membranes with a similar structure, which act as dividers between two subcellular compartments (e.g., membranes surrounding intracellular organelles) (Coskun and Simons, 2011).

The study of cell membrane properties in a living cell model is challenging because membrane properties and functions are determined by multiple cellular mechanisms and a complex biological structure. As a result, simplified cell models are often used in research. Liposomes ranging in size from 1–200 μm are also called giant unilamellar vesicles (GUVs) and are commonly used cell substitute. They consist of a bilayer of phospholipids which start to form lipid vesicles as a result of hydrophobic interactions in aqueous solution (Seifert, 1997; Safran, 1999; Nagle, 2013; Dimova, 2014). Due to their appropriate size, they can be directly observed under a light phase-contrast microscope. Typically, researchers isolate a vesicle or a small group, expose it to a change in its microenvironment and observe the differences over an extended period of time (Zupanc and Drobne, 2011; Stein et al., 2017; Penič et al., 2020).

### 1.5. *Preparation of phospholipid vesicles*

Several approaches are used to prepare giant unilamellar phospholipid vesicles, such as lipid film hydration, electroformation, lipid emulsification, microfluidic methods (hydrodynamic flow directing) and others (Pereno et al., 2017). A more detailed review of vesicle preparation methods is described in (Walde et al., 2010).

Electroformation is the most commonly used process for preparing vesicles from phospholipid components in the laboratory as it is fast (1–3 hours on average) and efficient. The method was first reported in 1986 (Angelova and Dimitrov, 1986). The proposed electroformation process is carried out by applying a lipid solution to a platinum electrode. When the solvent evaporates, a lipid film forms on the electrodes. The electrodes are transferred into an aqueous solution of sugar or salt and the alternating electrical field between the electrodes stimulates and accelerates the lipid vesicle formation process.

### 1.6. *Voltage source for electroformation*

A function generator is most commonly used as a sine wave AC voltage source. The electroformation of vesicles takes place in a weak electrical field (< 100 V/m), which, given the small distance between the electrodes, can be obtained at low voltages (1–5 V) and harmonically alternates with frequencies in the range of 1–10 Hz, allowing simpler, cheaper and smaller devices to be used for the process. That is why we chose to substitute the function generator with an affordable open-source AC voltage source alternative as a great example of applying Open Science principles.

We used a microcontroller and a filtered PWM (pulse-width modulated) digital output. This allows for the complete electroforming protocol to be stored in the memory of the microcontroller and triggered at the push of a button and thus eliminating repetitive action of needing to configure the source.

An important part of this research involves the prototyping of the AC generator and the evaluation of the device's performance. The circuit was built around the open-source Arduino microcontroller system. The circuit and the source code have been released under the MIT open-source license to promote wider adoption. In order to make the final product small and thus portable, easy to use and fully powered via the USB, we used only readily available and low-cost electronic components. The documentation for making the AC source for electroformation is available on GitHub (https://github.com/umalavasic/electroformation).

## 2. GUV electroformation protocol

The protocol for the preparation of giant unilamellar vesicles is determined in advance to achieve reproducibility of the experimental results. In this section we will focus only on the part of the protocol that is directly related to the source of the electrical voltage between the electrodes, irrespective of the choice of lipids, solutions, type of electrodes and other factors.

The standard electroformation protocol consists of three steps. In the first phase, the amplitude of the electrical field strength between the electrodes is gradually increased to a maximum value (typically $E_{max}$ < 100 V/m). This phase is followed by an optional vesicle swelling phase, where the field amplitude is maintained at the maximum value for a certain time interval. This is followed by a final phase where we typically reduce the frequency and stimulate the phospholipid bilayer to assemble into spherical vesicles (Méléard et al., 2009). The shape of the electrodes, and particularly their distance from each other, are important factors in determining the voltage of the generator in order to achieve the correct electrical field strength in the solution. Voltage between the electrodes, frequency and duration of each phase can have an overall effect on the size and mechanical properties of the formed vesicles (Drabik et al., 2018).

The operating protocol can also be modified accordingly by adding or substituting different reagents, such as using a salt solution instead of sugar (Méléard et al., 2009). In applied research, the first two operational phases are usually combined into one (Santhosh et al., 2020).

In routine laboratory experiments, the electroformation process is most often carried out in aqueous glucose solutions. Our work follows the key steps described by Stein et al. (Stein et al., 2017). The two platinum wire electrodes with a diameter of ≈ 2 mm are spaced 5 mm apart. The operating parameters of the vesicle electroformation system are also given in **Figure 2**.

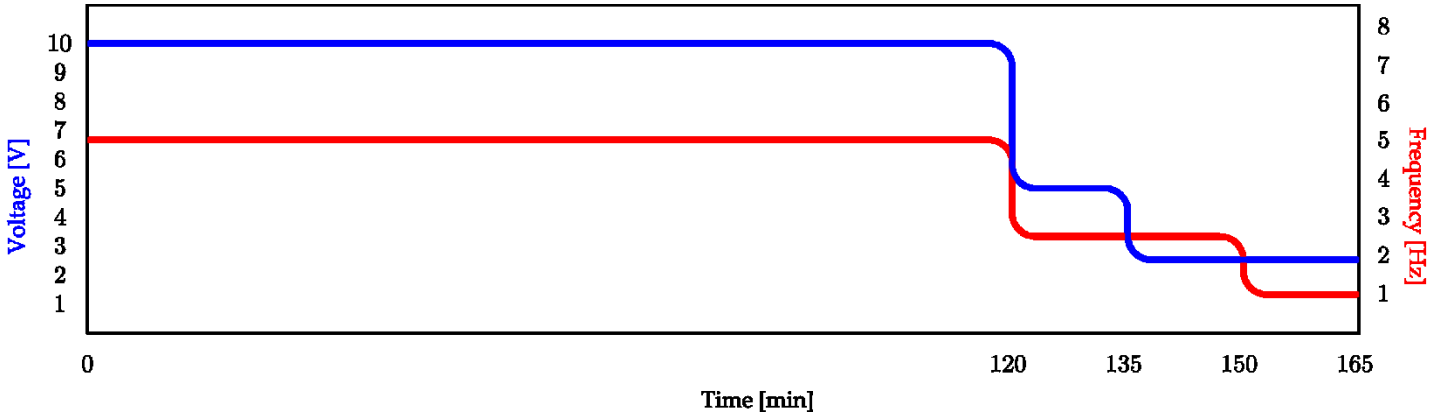



**Figure 2**. Typical electroformation protocol parameters. Red line describes time variance of frequency, and blue line the voltage of sinusoidal signal.

A gradual decrease in the frequency and voltage of the electrical field causes the swollen lipid vesicles to slowly detach from the surface of the electrodes and remain free floating in solution.

## 3. Generator circuit description

Using the Arduino Uno microcontroller platform, we generated a PWM signal with a frequency of 31,4 kHz. The duty cycle was determined based on a pre-calculated table of sine values. The signal was filtered with a first-order low-pass filter that attenuated frequencies above 150 Hz to obtain the desired sinusoidal signal. The signal manipulation circuit is shown in **Figure 3**.

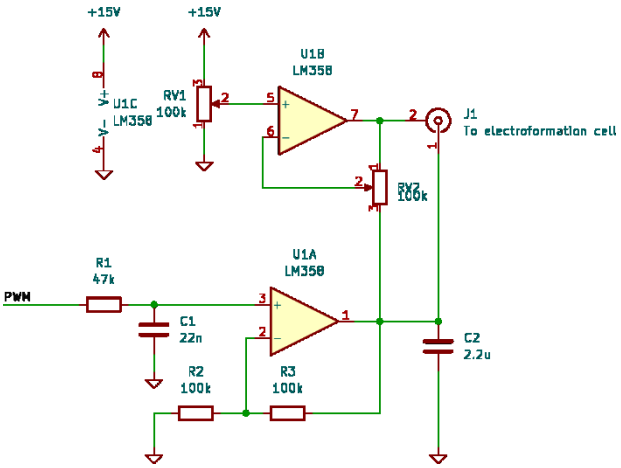
The output of the low-pass filter is connected to the non-inverting input of the operational amplifier, which has been set to a gain of a factor of 2. The operational amplifier used is a Texas Instruments LM358, which has two units in a DIP-8 housing. The second unit was used as an inverting operational amplifier. Potentiometer RV1 was used to set the gain to a factor of 2,5 while potentiometer RV2 was used to set the offset voltage of the operational amplifier to prevent it from being saturated by the signal.

Due to the single power supply of the operational amplifier, the electrodes were connected differentially between the outputs of the two operational amplifiers.

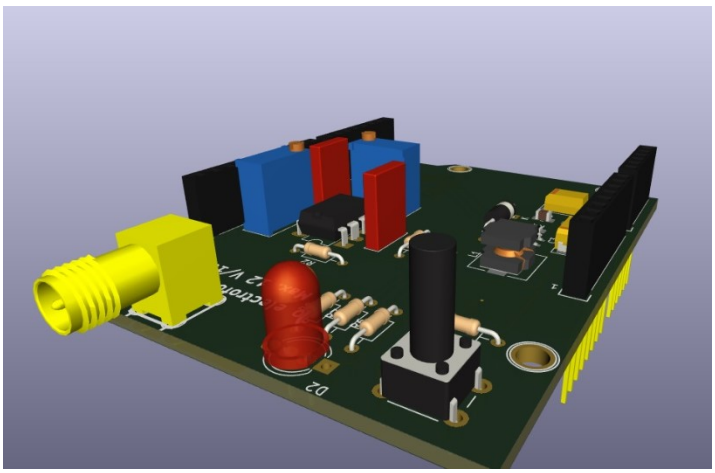
The calibration was performed using an oscilloscope. First, we set the gain with potentiometer RV1 so that when the maximum amplitude of the microcontroller output was set, we got a peak voltage of 10 $V_{pp}$ between the output terminals, followed by cancellation of the DC component of the voltage with potentiometer RV2.

The operational amplifier was powered with a voltage of 15 V, which was obtained from the supply voltage of the Arduino microcontroller system using a boost converter. We used a module based on the SX1308 converter from Sunrom.

The maximum achievable frequency at the output is limited to about 150 Hz and the voltage to 10 $V_{pp}$. The current capacity of the output is limited by the characteristic of the selected operational amplifier which is ≈ 40 mA. This does not pose a practical problem for a high-ohm load represented by a vial with electrodes and solution.




(a)



(b)

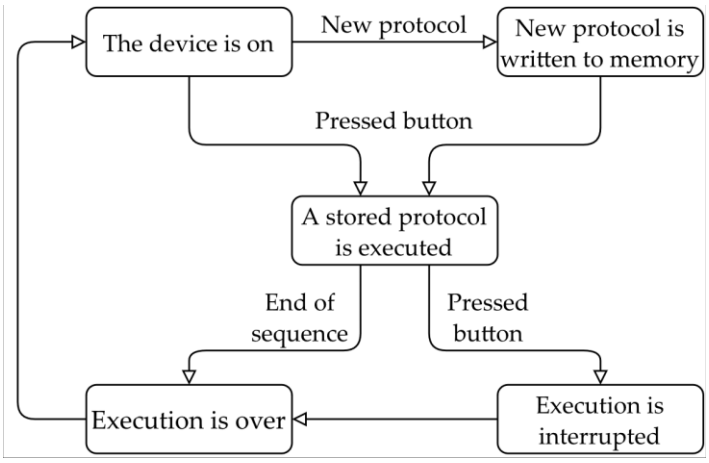
**Figure 3**. Circuit diagram (a) and a 3D graphic showing the board (b).

## 4. Code execution flow

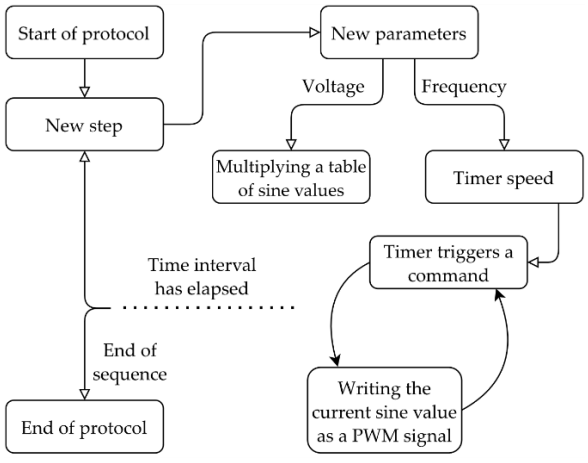
The parameters for the execution of the vesicle preparation protocol (**Figure 4**) are stored on the device in an electrically erasable programmable read-only memory (EEPROM). After switching on the device, the colored LED lights up and turns light blue.

Pressing the button starts the stored protocol and the LED turns purple. When the protocol is finished, the LED turns green. The protocol execution can be aborted during operation by pressing the button and the LED will turn red. A flow chart of the device operation is presented in Figure 4a.

The last used protocol is persistently stored in the device's memory and can be edited by connecting the device via USB to a computer and using an application developed in Python. The device receives the new instruction for protocol execution via the serial port at a baud rate of 115 200 Bd. The instruction consists of comma-separated electric field parameters, with the individual steps separated by semicolons. The beginning and the end of the instruction are delimited by the characters "<" and ">". When accepted by the device, the old protocol is overwritten by the new one and the LED turns dark blue. A schematic of the implementation of the electroformation protocol is shown in Figure 4b.




(a)




(b)

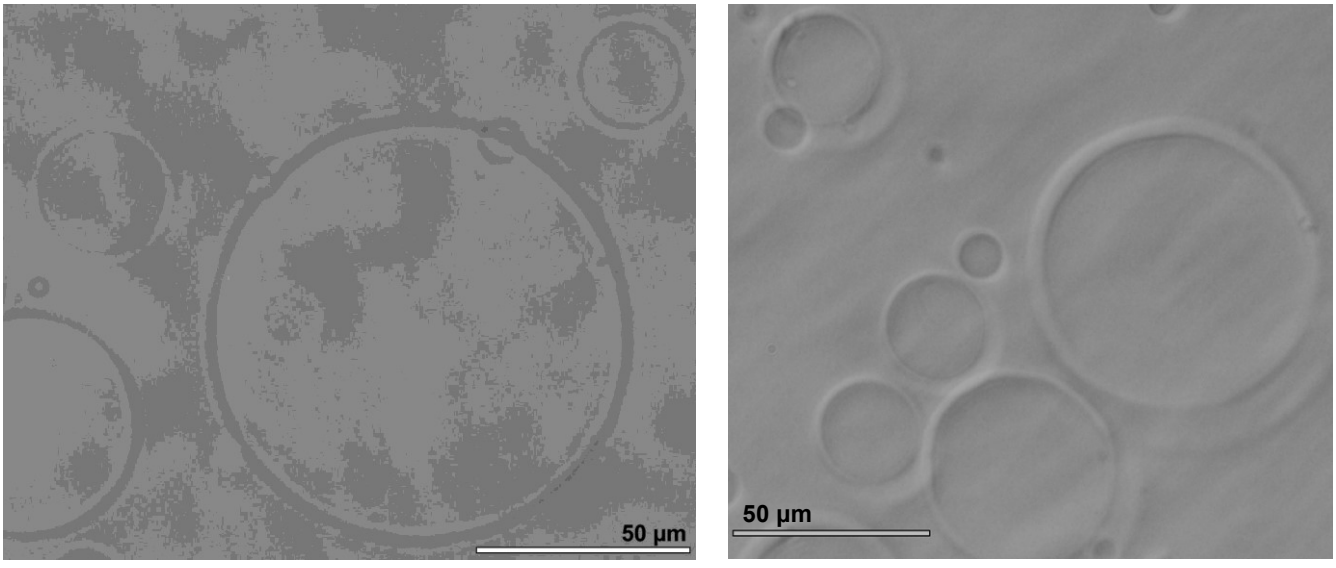
**Figure 4**. Flow chart of the device's operation (a) and of the electroformation protocol (b).

## 5. Results and conclusion

The harmonic voltage source was tested in the laboratory, following the preparation instructions of reference (Stein et al., 2017). Giant phospholipid vesicles were first prepared by electroformation from the synthetic lipid POPC (1-palmitoyl-2-oleoyl-*sn*- glycero-3-phosphocholine), and then from its mixture with cholesterol (both from Avanti Polar Lipids). On each platinum wire electrode, 20 μL of the above lipid diluted in a 2:1 mixture of chloroform and methanol to a concentration of 1 mg/mL was applied by drop creep to the electrodes and dried in a desiccator for 45 min at constant vacuum. After drying, the electrodes were placed in a 2 mL plastic vial containing 1,8 mL of 0,3 M sucrose solution.

The electrodes were connected to the terminals of the harmonic voltage source and the procedure was started according to the protocol described in Section 2. After a total of 165 min, the electrodes were disconnected and the contents of the vial were carefully transferred into 3,60 mL of 0,3 M glucose solution (the volume ratio of sucrose to glucose in the final mixture was 1:2). 200 μL of the suspension was pipetted into experimental chambers (manufactured by Grace Bio-Labs) which allow quantitative field analysis and uniform image capture under the microscope. The chambers were placed under the microscope and left for 30 minutes to allow the contents to settle to the bottom. A light microscope (Nikon Eclipse TE2000-S) was used to examine the samples using immersion oil at 100x magnification.

Samples of giant phospholipid vesicles from both formations were captured with a microscope camera (model UI-3280CP from IDS Imaging). Figure 5a shows a vesicle whose membrane consists only of POPC phospholipid without added cholesterol, while Figure 5b shows a set of vesicles whose membranes contain a mixture of the two. The addition of cholesterol also results in a slightly thicker membrane, which is nicely highlighted by microscopy.




(a) (b)

**Figure 5**. A 100x magnification image of giant phospholipid vesicles. The images show membranes with POPC phospholipid (a) and its 1:4 mixture with cholesterol (b).

The amounts of vesicles formed in both formation cases were comparable to those obtained using a commercial function generator. The electroforming process does not require an exact sinusoidal waveform, so the signal construction using PWM modulation

is sufficient. The protocols reported in the literature even mention other signal waveforms. In his review article, Méléard mentions the use of an alternating source of either rectangular or sinusoidal form (Méléard et al., 2009), and under specific conditions certain authors (Breton et al., 2015) use protocols that combine both signals – sinusoidal and rectangular.

The presented sinusoidal voltage generator of adjustable amplitude and frequency represents a portable and inexpensive solution for the electroformation of giant unilamellar vesicles. An additional advantage is the complete automation of the electroformation process and the ease of use of the generator, since the protocol steps are stored in memory and executed automatically in sequence, so that the device does not need to be reconfigured during the execution of the protocol. The generator does not need any additional power supply, as it can be operated using the USB output of the computer. An additional benefit of the connection to the computer is the possibility to implement communication with the computer and to be informed via the Internet connection about the current status of the protocol execution. In order to make the generator available to a wider community, we have released the design and the software code under the MIT open-source license.

We conclude that our research stands as an example of importance in practice of Open Science. By publishing source code and documenting the design openly and freely we simplify the work of other researchers and provide them with an affordable and open-source replacement for an expensive device. That not only makes their job easier but also, through collaboration, improves code quality as potential bugs can be identified earlier in the process. The code can also be modified to be applied for other projects that require voltage generation with similar parameters.

**Funding:** This research was supported by European Union's Horizon 2020 research and innovation program under grant agreement No 801338 (Ves4Us), and by Slovenian Research Agency through the core foundlings No P3-0388 & P2-0244, and project No L3-2621.

**Conflicts of Interest:** The authors declare no conflict of interest.

**References**

1. Angelova MI, Dimitrov DS. Liposome electroformation. Faraday discussions of the Chemical Society. 1986; 81: 303–311. DOI:10.1039/dc9868100303
2. Breton M, Amirkavei M, Mir LM. Optimization of the electroformation of giant unilamellar vesicles (GUVs) with unsaturated phospholipids. The journal of membrane biology. 2015; 248: 827–835. DOI: 10.1007/s00232-015-9828-3
3. Cevc G, Marsh D. Phospholipid bilayers: physical principles and models. Wiley. 1987.
4. Coskun Ü, Simons K. Cell membranes: the lipid perspective. Structure. 2011; 19: 1543–1548. DOI: 10.1016/j.str.2011.10.010
5. Dimova R. Recent developments in the field of bending rigidity measurements on membranes. Adv. Coll. Interf. Sci. 2014; 208: 225–234. DOI: 10.1016/j.cis.2014.03.003
6. Drabik D, Doskocz J, Przybyło M. Effects of electroformation protocol parameters on quality of homogeneous GUV populations. Chemistry and physics of lipids. 2018; 212: 88–95. DOI: 10.1016/j.chemphyslip.2018.01.001
7. Gazvoda de Reggi M, Malavašič U, Jeran M, et al. Generator izmenične napetosti za elektroformacijo orjaških fosfolipidnih veziklov. In: Žemva A, Trost A, editors. Zbornik tridesete mednarodne Elektrotehniške in računalniške konference. ERK Portorož, Društvo Slovenska sekcija IEEE. 2021; pp. 419-422.
8. Méléard P, Bagatolli LA, Pott T. Giant unilamellar vesicle electroformation: From lipid mixtures to native membranes under physiological conditions. Methods in enzymology. 2009; 465: 161–176. DOI: 10.1016/S0076-6879(09)65009-6
9. Nagle JF. Introductory Lecture: Basic quantities in model biomembranes. Faraday Discuss. 2013; 161: 11–29. DOI: 10.1039/C2FD20121F.
10. Penič S, Mesarec L, Fošnarič M, et al. Budding and fission of membrane vesicles: A mini review. Frontiers in Physics. 2020; 8: 342. DOI: 10.3389/fphy.2020.00342
11. Pereno V, Carugo D, Bau L, et al. Electroformation of Giant Unilamellar Vesicles on Stainless Steel Electrodes. ACS Omega. 2017; 2: 994–1002. DOI: 10.1021/acsomega.6b00395
12. Safran SA. Curvature elasticity of thin films. Adv. Phys. 1999; 48: 395–448. DOI: 10.1080/000187399243428.
13. Santhosh P, Genova J, Iglic A, et al. Influence of cholesterol on bilayer fluidity and size distribution of liposomes. Comptes rendus de l'Académie bulgare des Sciences. 2020; 73. DOI:10.7546/crabs.2020.07.07

14. Seifert U. Configurations of fluid membranes and vesicles. Adv. Phys. 1997; 46: 13–137. DOI: 10.1080/00018739700101488.
15. Stein H, Spindler S, Bonakdar N, et al. Production of isolated giant unilamellar vesicles under high salt concentrations. Frontiers in physiology. 2017; 8: 63. DOI: 10.3389/fphys.2017.00063
16. Vicente-Saez R, Martinez-Fuentes C. Open Science now: A systematic literature review for an integrated definition. Journal of Business Research. 2018; 88: 428–436. DOI: https://doi.org/10.1016/j.jbusres.2017.12.043.
17. Walde P, Cosentino K, Engel H, et al. Giant vesicles: preparations and applications. ChemBioChem. 2010; 11: 848–865. DOI: 10.1002/cbic.201000010
18. Wikipedia contributors. Celična membrana – Wikipedia, The Free Encyclopedia. Accessed 20 July 2021 Available from https://sl.wikipedia.org/wiki/Celična_membrana.
19. Wilson G, Aruliah DA, Brown CT, et al. Best practices for scientific computing. PLoS Biol. 2014; 12: e1001745. DOI: 10.1371/journal.pbio.1001745.
20. Zupanc J, Drobne D. Populacije orjaških lipidnih veziklov kot model za studij bio-nano interakcij/Giant Lipid Vesicle Populations as a Model for Bio-nano Interaction Studies. Informatica Medica Slovenica. 2011; 16: 1-12.
21. What is open source? Opensource.com. Accessed 31 December 2021a Available from https://opensource.com/resources/what-open-source.
22. What is open hardware? Opensource.com. Accessed 31 December 2021b Available from https://opensource.com/resources/what-open-hardware.

**Appendix A:** Firmware for Arduino

```
// WAVEFORM
#define WAVEFORM_PIN              11            // Don't change unless you also adjust timers and PWM
                                                // frequency accordingly
#define SIGNAL_VOLTAGE_AMPLITUDE  5             // [V]
#define PROTOCOL_MAX_STAGES       10            // Currently set to accept 10 stages in single protocol

// BUTTON
#define BUTTON_IS_USED            true
#define BUTTON_PIN                2
#define BUTTON_PIN_MODE           INPUT_PULLUP  // change to INPUT when using external resistors

// RGB LED
#define RGB_LED_IS_USED           true
#define RED_LIGHT_PIN             3
#define GREEN_LIGHT_PIN           6
#define BLUE_LIGHT_PIN            5

// TIMER
#define USE_TIMER_1               true
#include <TimerInterrupt.h>
#include <ISR_Timer.h>

// EEPROM
#include <EEPROM.h>

// MARK: - Signal Generation
const byte sineWave[] = {
  128, 131, 134, 137, 140, 143, 146, 149, 152, 155, 158, 162, 165, 167, 170, 173, 176, 179, 182, 185, 188,
         190, 193, 196, 198, 201, 203, 206, 208, 211, 213, 215, 218, 220, 222, 224, 226, 228, 230, 232, 234,
         235, 237, 238, 240, 241, 243, 244, 245, 246, 248, 249, 250, 250, 251, 252, 253, 253, 254, 254, 254,
         255, 255, 255, 255, 255, 255, 255, 254, 254, 254, 253, 253, 252, 251, 250, 250, 249, 248, 246, 245,
         244, 243, 241, 240, 238, 237, 235, 234, 232, 230, 228, 226, 224, 222, 220, 218, 215, 213, 211, 208,
         206, 203, 201, 198, 196, 193, 190, 188, 185, 182, 179, 176, 173, 170, 167, 165, 162, 158, 155, 152,
         149, 146, 143, 140, 137, 134, 131, 128, 124, 121, 118, 115, 112, 109, 106, 103, 100, 97, 93, 90, 88,
         85, 82, 79, 76, 73, 70, 67, 65, 62, 59, 57, 54, 52, 49, 47, 44, 42, 40, 37, 35, 33, 31, 29, 27, 25,
         23, 21, 20, 18, 17, 15, 14, 12, 11, 10, 9, 7, 6, 5, 5, 4, 3, 2, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
         1, 1, 2, 2, 3, 4, 5, 5, 6, 7, 9, 10, 11, 12, 14, 15, 17, 18, 20, 21, 23, 25, 27, 29, 31, 33, 35, 37,
         40, 42, 44, 47, 49, 52, 54, 57, 59, 62, 65, 67, 70, 73, 76, 79, 82, 85, 88, 90, 93, 97, 100, 103,
         106, 109, 112, 115, 118, 121, 124
  // https://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml
};
```

```
const unsigned waveformResolution = sizeof(sineWave) / sizeof(sineWave[0]); // = 256
byte waveform[waveformResolution];
unsigned waveformPosition;

// MARK: - Protocol
struct Protocol {
  unsigned long duration; // [s]
  float frequency;        // [Hz], 0.0 < float frequency
  float amplitude;        // [V],  0.0 < float amplitude < SIGNAL_VOLTAGE_AMPLITUDE
};
Protocol protocol[PROTOCOL_MAX_STAGES];
byte protocolStages;
bool protocolInProgress = false;

// MARK: - Serial Communication
const unsigned messageSize = 512; // Maximum amount of characters in one serial port message
char receivedMessage[messageSize];
bool receivedProtocol = false;

// MARK: - Button
byte buttonState = HIGH, lastButtonState = HIGH;
unsigned long lastDebounceTime, debounceDelay = 50; // debounce time; increase if the output flickers

// MARK: - RGB LED
struct Color {
  byte red, green, blue;
};

const struct Colors {
  Color red = {255, 0, 0};
  Color green = {0, 255, 0};
  Color blue = {0, 0, 255};
  Color cyan = {0, 255, 255};
  Color magenta = {255, 0, 255};
  Color yellow = {255, 75, 0};
  Color white = {255, 255, 255};
} colors;

// MARK: - Life Cycle
void setup() {
  setupPWM();
  setupTimer();
  setupSerial();
  setupButton();
  setupLED();

  loadProtocol();
  exportProtocol();

  setRGBColor(colors.cyan);
}

void loop() {
  if (!protocolInProgress) {
    receiveProtocol();
    if (receivedProtocol) {
      receivedProtocolMessage();
      receivedProtocol = false;
    }
  }

  if (BUTTON_IS_USED) {
    handleButton();
  }
}

// MARK: - Setup Functions
```

```
void setupPWM() {
  pinMode(WAVEFORM_PIN, OUTPUT);
  // Set Pin 11 (WAVEFORM_PIN) PWM frequency to 31372.55 Hz, instead of default 490.20 Hz.
  TCCR2B = TCCR2B & B11111000 | B00000001;
}

void setupTimer() {
  ITimer1.init();
}

void setupSerial() {
  Serial.begin(115200);
  Serial.print("READY");
}

// MARK: - RGB LED
void setupLED() {
  if (RGB_LED_IS_USED) {
    pinMode(RED_LIGHT_PIN, OUTPUT);
    pinMode(GREEN_LIGHT_PIN, OUTPUT);
    pinMode(BLUE_LIGHT_PIN, OUTPUT);
  }
}

void setRGBColor(Color color) {
  if (RGB_LED_IS_USED) {
    analogWrite(RED_LIGHT_PIN, color.red);
    analogWrite(GREEN_LIGHT_PIN, color.green);
    analogWrite(BLUE_LIGHT_PIN, color.blue);
  }
}

// MARK: - Button
void setupButton() {
  if (BUTTON_IS_USED) {
    pinMode(BUTTON_PIN, BUTTON_PIN_MODE);
  }
}

void handleButton() {
  int reading = digitalRead(BUTTON_PIN);
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    byte state = buttonState;
    buttonState = reading;
    if (reading != state && reading == HIGH) {
      buttonPressed();
    }
  }
  lastButtonState = reading;
}

void buttonPressed() {
  Serial.println("Button pressed");
  if (protocolInProgress) {
    cancelProtocol();
  } else {
    runProtocol();
  }
}

// MARK: - Serial Communication Functions
void receiveProtocol() {
  static bool receiving = false;
  static unsigned index;
```

```
  char startMarker = '<', endMarker = '>', receivedByte;
  while (Serial.available() > 0 && !receivedProtocol) {
    receivedByte = Serial.read();

    if (receiving) {
      if (receivedByte != endMarker) {
        receivedMessage[index] = receivedByte;
        index++;
        if (index >= messageSize) {
          index = messageSize - 1;
        }
      } else {
        receivedMessage[index] = '\0';
        receiving = false;
        index = 0;
        receivedProtocol = true;
      }
    } else if (receivedByte == startMarker) {
      receiving = true;
    }
  }
}

void receivedProtocolMessage() {
  parseProtocol();
  exportProtocol();
  saveProtocol();
  setRGBColor(colors.blue);
  if (!BUTTON_IS_USED) {
    runProtocol();
  }
}

void parseProtocol() {
  char * token = receivedMessage;
  unsigned int stages = 0;
  for (int i = 0; token; i++) {
    token = (i == 0) ? strtok(receivedMessage, ",") : strtok(NULL, ",");
    if (!token) {
      // When the token == NULL, we've reached the end of data.
      continue;
    }
    unsigned long duration = atol(token);
    token = strtok(NULL, ",");
    float frequency = atof(token);
    token = strtok(NULL, ";");
    float amplitude = atof(token);

    protocol[i] = { duration, frequency, amplitude };
    stages++;
  }
  protocolStages = stages;
}

void exportProtocol() {
  Serial.print("<");
  for (int i = 0; i < protocolStages; i++) {
    unsigned long duration = protocol[i].duration;
    float frequency = protocol[i].frequency;
    float amplitude = protocol[i].amplitude;

    if (i != 0) {
      Serial.print(";");
    }
    Serial.print(duration);
    Serial.print(",");
    Serial.print(frequency);
```

```
    Serial.print(",");
    Serial.print(amplitude);
  }
  Serial.print(">");
}

// MARK: - EEPROM Functions
void saveProtocol() {
  EEPROM.write(0, protocolStages); // Save number of stages to address 0

  for (int i = 0; i < protocolStages; i++) {
    Protocol stage = protocol[i];

    unsigned long duration = stage.duration;
    float frequency = stage.frequency;
    float amplitude = stage.amplitude;

    int durationAddress = (sizeof(duration) + sizeof(frequency) + sizeof(amplitude)) * i + 1;
    int frequencyAddress = durationAddress + sizeof(duration);
    int amplitudeAddress = frequencyAddress + sizeof(frequency);

    writeUnsignedLong(duration, durationAddress);
    writeFloat(frequency, frequencyAddress);
    writeFloat(amplitude, amplitudeAddress);
  }
}

void loadProtocol() {
  protocolStages = EEPROM.read(0);

  for (int i = 0; i < protocolStages; i++) {
    unsigned long duration;
    float frequency;
    float amplitude;

    int durationAddress = (sizeof(duration) + sizeof(frequency) + sizeof(amplitude)) * i + 1;
    int frequencyAddress = durationAddress + sizeof(duration);
    int amplitudeAddress = frequencyAddress + sizeof(frequency);

    duration = readUnsignedLong(durationAddress);
    frequency = readFloat(frequencyAddress);
    amplitude = readFloat(amplitudeAddress);

    protocol[i] = { duration, frequency, amplitude };
  }
}

void writeUnsignedLong(unsigned long value, int startAddress) {
  byte* bytes = (byte*) &value;
  for (int i = 0; i < sizeof(unsigned long); i++) {
    EEPROM.write(startAddress + i, bytes[i]);
  }
}

unsigned long readUnsignedLong(int startAddress) {
  const byte valueSize = sizeof(unsigned long);
  union {
    byte bytes[valueSize];
    unsigned long value;
  } valueUnion;
  for (int i = 0; i < valueSize; i++) {
    valueUnion.bytes[i] = EEPROM.read(startAddress + i);
  }
  return valueUnion.value;
}

void writeFloat(float value, int startAddress) {
```

```
  byte* bytes = (byte*) &value;
  for (int i = 0; i < sizeof(float); i++) {
    EEPROM.write(startAddress + i, bytes[i]);
  }
}

float readFloat(int startAddress) {
  const byte valueSize = sizeof(float);
  union {
    byte bytes[valueSize];
    float value;
  } valueUnion;
  for (int i = 0; i < valueSize; i++) {
    valueUnion.bytes[i] = EEPROM.read(startAddress + i);
  }
  return valueUnion.value;
}

// MARK: - Protocol Execution
void runProtocol() {
  protocolInProgress = true;
  setRGBColor(colors.magenta);
  for (int i = 0; i < protocolStages; i++) {
    if (!protocolInProgress) {
      return; // Protocol has been canceled.
    }
    Protocol stage = protocol[i];
    runStage(stage);
  }
  protocolFinished(true);
}

void runStage(Protocol stage) {
  unsigned long duration = stage.duration;
  float frequency = stage.frequency;
  float amplitude = stage.amplitude;

  // 1. Populate 'waveform' array with sine wave values modulated with amplitude.
  float scaler = amplitude / SIGNAL_VOLTAGE_AMPLITUDE;
  for (int i = 0; i < waveformResolution; i++) {
    waveform[i] = byte(round(sineWave[i] * scaler));
  }

  // 2. Start the timer interrupt process.
  float timerFrequency = frequency * 2 * waveformResolution;
  ITimer1.attachInterrupt(timerFrequency, tick);

  // 3. Wait for the duration of the sequence, then stop the timer interrupt process.
  unsigned long startTime = millis(), currentTime = millis();
  while (currentTime - startTime < duration * 1000) {
    if (BUTTON_IS_USED) {
      handleButton();
    }
    currentTime = millis();
  }

  ITimer1.disableTimer();
}

void tick() {
  byte instantaneousValue = waveform[waveformPosition];
  analogWrite(WAVEFORM_PIN, instantaneousValue);
  waveformPosition = (waveformPosition + 1) % waveformResolution;
  // Serial.println(instantaneousValue); -- to check sinewave using serial plotter
}

void cancelProtocol() {
```

```
  ITimer1.disableTimer();
  waveformPosition = 0;
  protocolFinished(false);
}

void protocolFinished(bool finished) {
  protocolInProgress = false;
  if (finished) {
    setRGBColor(colors.green);
  }
  else {
    setRGBColor(colors.red);
  }
  analogWrite(WAVEFORM_PIN, 0);
}
```