Call Activation When Detecting Cardiac Irregularities

Tanisha Rajgor, Archita Nemalikanti
Hopkinton High School

I. Abstract

About 610,000 people die every year in the United States due to heart-related diseases. Many of those cases occur because of a lack of proper medical attention in time. Paramedics do not arrive in time when a person is struck immediately with a cardiovascular disease. There are many devices on the market that can measure one's heart rate, such as the Apple Watch and Fitbit. However, when a person has an irregular heart rate these devices can only inform the victim of it. The victim must call for medical on their own. But what if the person cannot physically call, and there is no one to help them?

When heart rate drastically drops or raises, a device created should be able to call 911. There are four modules to the project: the signal (pulse) generator, the Arduino board, the Bluetooth module, and the app in the phone. A signal generator simulates a heart rate through electronic signals or beats. The signal generator is an analog device that sends out a pulse to the Arduino. The Arduino, a digital device, allows the circuit to flow, and is the 'brain' of the circuit. The Arduino is supposed to receive the heart rate from the signal generator and display the heart rate on the display (which is located on the breadboard). If the heart rate is over 80 bpm or less than 50 bpm, then the Arduino should begin a countdown of thirty seconds on the display. When the countdown hits zero, the Arduino should recognize it, and send a signal to the Bluetooth module located on the breadboard. The Bluetooth Module should then send a message to the app in the phone, and then the phone should be able to call 911 (or any other number, in this project, 911 should not actually be called, so it was replaced with a different number).

II. Introduction:

A device will be designed to simulate and analyze a human body's pulse (heart rat)e and when it is irregular, call for help inform others about the issue. This can be very helpful just in case the pulse increases arbitrarily- this can be very dangerous, as it can cause kidney failure immediately as the blood pressure goes drastically up or down. This is also very important as the victim experiencing an irregularity in heart rate, may not physically be able to call 911 by themselves, and there may also be no one else to help the victim. Hence, the device should also be able to call 911 when it detects an abnormality in the body automatically.

For the hardware of the project, we may use Arduino, a computer board where one can connect external components and code circuits using the C language and create projects. In order to call 911, a Bluetooth receiver on the Arduino board could be used to connect it to a phone. One of the parts that can be used as the Bluetooth receiver is the Arduino Bluetooth module. The Bluetooth module consists of four different ports: VCC (power), GND (ground), TX (transmitter), and RX (receiver). Each of these ports will be connected to the physical Arduino computer boards' ports with color-coordinated wires. Whenever the device detects an abnormality in the body, it could send a signal to the phone via Bluetooth and an app could make the phone call 911. The app could be created using the MIT App Inventor, which works with Android devices.

There are many devices on the market that are capable of measuring the heart rate of a person. The Apple Watch by Apple, for example, can measure one's heart rate and has saved the lives of a numerous amount of people. It and many other heart rate measures on the market measure heart rate by using green LED lights with light-sensitive photodiodes to detect the amount of blood flowing through one's wrist. When the heart beats the green light absorption is greater, and between beats, it is less. By flashing its LED very fast the watches can calculate a person's heart rate. Fitbit watches can also measure heart rate in a similar way.

The heart consists of four separate chambers that are used to pump blood to the body, two are on the left side of the heart and the other two chambers are on the right of the heart. On the right side, the right atrium receives blood that is deprived of oxygen used by the body and pumps the blood to the right ventricle. The right ventricle then pumps the blood to the lungs, which gets

rid of the carbon dioxides produced by the body and takes in oxygen. On the left side of the heart, the left atrium receives the oxygen-rich blood from the lungs and pumps the blood to the left ventricle. The left ventricle then pumps out the blood to the rest of the body through the aortic valve. This process repeats in a cycle at a regular rhythm, although sometimes this rate does change.

As due to regulation, the project is not allowed to have human testing, in the way of measuring the heart rate of the person. There would have to be a way to simulate the action of how the heart works to pump blood throughout the body in beats. To get around, this a signal generator, or something similar will be used. “A signal generator produces an electrical signal in the form of a wave.” (electric-notes). In the way that a heart sends out beats at a constant rate, a signal generator send out electric signals at a constant rate. Preferably, the two things that the signal generator should have the capacity to do are: being able to code how long the interval is between beats and being able to connect the signal generator to the Arduino board. The signal generator may be able to connect to the Arduino by Bluetooth, just like the phone. Or if the generator was like a chip, it may be able to be wired to the Arduino. There are also ways, to not use a physical generator, but to code a generator using the Arduino software. Many signal generators may have fixed values that may be adjustable, but many signal generators have a very limited amount of options. The goal is to be able to set the signal generator’s time intervals between pulses, similar to that of a human heart, maybe not exactly number-wise, but proportionally.

In order to make the demographic more widespread, instead of having just one point that the simulated heart rate would exceed over, there would be two benchmarks. Depending on the type of heart irregularity being worked on, the first benchmark would be a little high risk circumstance of the condition, not life threatening but, a little abnormal. The other benchmark, would represent a much more extreme and life-threatening circumstance, one that would need immediate medical attention. There would be an display on the Arduino that would execute this, and also show some basic and quick statistics of the heart rate. When the first benchmark has been passed, the display will ask the user “Is everything okay?” and then the user would have a choice to decide to call medical professionals or not, perhaps by a push button. Sometimes even

a person with no heart conditions or weakness, can get an abnormal heart rate by over exercising, being sick, or even being over stressed, so giving this option would give the user more control. But, in case something is an issue, or for example a person normally does have a low heart rate than average, after 30 seconds of display the warning message, the device will call 911 or a doctor, which the user can set. For the second benchmark, the device will automatically call 911, as this benchmark is life-threatening.

In this project, a resting heart-rate will be looked upon. The normal resting heart rate for adults is between 60-100 bpm (beats per minute). A resting heart rate above 100 bpm (tachycardia), or below 60 bpm (bradycardia) should be consulted by a physician. These rates would be examples of what the first benchmarks would be like. Different kinds of "Arrhythmia" or irregular heartbeats include: tachycardia (speeding heart rate), bradycardia (slow heart rate), Premature Ventricular Contractions (skipped heart beats). There are many different kinds of Arrhythmia, but most of them are different variations of the following conditions, which may happen or be caused by different parts of the heart. For this project, the complete Arduino circuit should be able to simulate and respond to the three types of heart irregularities mention above. In conclusion, the device should be able to respond to a cardiac irregularity, in the form of calling the paramedics, and notifying others and the user for help.

III. Methods

Code the program for the Arduino board being able to detect the different time intervals, at which the pulses are sent out by the signal generator. There should be two different points, or benchmarks that the device should look out for. One would be abnormally high, and the other life-threatening. And when the device does detect a difference at a certain level, it should react in a different ways depending on what benchmark has been reached by the simulated heart rate. When reaching the first, lower, benchmark, the display on the device should present the statement “Is everything okay?”, and then give the user the option to call 911 or not. If the user does not respond after 30 seconds the device should automatically call 911. If the second benchmark has been reached, the device should be able to send a signal to the Bluetooth receiver, to send to an app on the phone.

Also code and program the app on the phone on MIT App Inventor, which the signal from the Bluetooth receiver should go to. Using the app, the phone will be able to call a number. The app should have a simple layout, and should show statistics about the simulated heart rate. The device should start out by having a normal simulated heart rate and then after 5 seconds, the device should start the irregular heart rate. First start with the first benchmark heart rate then the second benchmark heart rate. See if the device responds in the appropriate manner to the irregularity. Test this 5 times, for both the 1st and 2nd benchmarks.

The breadboard contains a coordinate-like grid which was referenced for locations of certain parts. Put the display on A1- A18 of the breadboard. Take a jumper wire and put it in E1 of the breadboard (LCD pin 1) and connect it to the ground of the breadboard (the blue line at the top with a minus). Take a jumper wire put it in E2 of the breadboard (LCD pin 2) and put it in the power supply of the breadboard (red line below the ground). Take a jumper wire and put it in E3 (pin 3 of LCD) of the breadboard and connect it to F22 of the breadboard. Take a jumper wire and put it in E4 of the breadboard (LCD pin 4) and connect it to pin 12 of the ARDUINO. Take a jumper wire and put it in E5 (pin 5 of LCD) of the breadboard and connect it to the ground of the breadboard. Take a jumper wire and put it in E6 of the breadboard (pin 6 of LCD) and connect it to pin 11 of the ARDUINO (there is a tilde in front if pin 11 of the Arduino). Take a jumper wire and put it in E11 of the breadboard (pin 11 of LCD) and connect it to pin 5 of the ARDUINO.

Take a jumper wire and put it in E12 of the breadboard (pin 12 of LCD) and connect it to pin 4 of the ARDUINO. Take a jumper wire and put it in E13 of the breadboard (pin 13 of LCD) and connect it to pin 3 of the ARDUINO. Take a jumper wire and put it in E14 of the breadboard (pin 14 of LCD) and connect it to pin 2 of the ARDUINO. Take a jumper wire and put it in E15 of the breadboard (pin 15 of LCD) and connect it to the ground of the BREADBOARD. Take a jumper wire and put it in E16 of the breadboard (pin 16 of LCD) and connect it to the ground of the breadboard. Take a potentiometer and put it on G20 through G23 of the breadboard. Take a jumper wire and put it in F21 of the breadboard, and connect it to the ground of the breadboard. Take a jumper wire and put it in F23 and connect it to the power supply of the breadboard. Take the Bluetooth Module and put it in pins J26 through J31. Take a jumper wire and put it in pin 25 of the power supply of the breadboard. Connect that jumper wire to 5V power section of the Arduino. Take a jumper wire and put it in pin 25 of the ground of the breadboard. Connect that jumper wire to the Ground in the digital PWM section of the Arduino. Take a jumper wire and put it in pin G27 of the breadboard (or VCC of Bluetooth module) and connect it to the 3.3V pin of the Arduino (in the power section). Take a jumper wire and put it in pin G28 of the breadboard (or GND of Bluetooth module) and connect it to the GND pin of the Arduino (in the power section). Take a jumper wire and put it in pin G29 of the breadboard (or TXD of module) and connect it to the RX (0) pin of the Arduino (in the digital section). Take a jumper wire and put it in pin G30 of the breadboard (or RXD of Bluetooth module) and connect it to the TX (1) pin of the Arduino (in the digital section). Put three pin headers in the PWM and GND sections of the Signal Generator. Connect the signal generator in pins G37 through G39. Take a jumper wire and put it in pin F27 of the breadboard (VCC pin of the Bluetooth module) and connect it to the VIN+ pin of the signal generator (uppermost pin of the section). Take a jumper wire and put it in pin J37 of the breadboard (PWM pin of the signal generator) and connect it to pin 6 or the Arduino (in the digital PWM section---it has a tilde). Take a jumper wire and put it in pin J38 of the breadboard (GND pin of the signal generator) and connect it to the ground of the breadboard. Take a jumper wire and put it in the ground of pin 45 of the breadboard. Connect it to the GND pin of the Arduino (power section of the Arduino).

The code must be written in the Arduino IDE (Integrated Development Environment), which is in the programming language C. Include all of the functions that are in the "LiquidCrystal.h" library.) Set up all of the parameters for the Liquid Crystal Display (the LCD is plugged into these ports). Pin 12 (Register Select) of the Arduino tells whether it is sending instruction or data. Pin 11 is turning on the LCD. The rest of the pins transport data/instruction. Then, set up a variable with integer value 30, 'counter'. This variable is for the 30-second countdown. Set the port the Signal Generator is connected into port 6, which is represented by the variable 'pwminput'. A signal generator is sending analog pulses to the Arduino in pin 6. The Pulse Width Modulation pins convert analog information into digital waves that the Arduino can read. Set the Boolean variable to false. Boolean is a variable that tells YES (1), or NO (0) it can only have 2 values. This value is false to begin with, but it will change.

Set up a variable called 'waitcounter' to 0, which will eventually count for 60 seconds. The Arduino has to wait for 60 seconds before it begins. If it does not wait for 60 secs, the display will automatically show the counter. Create the 'void setup' function, which will set certain commands before the main part of the program starts. Set PWM digital pin 6, to be the input for the Arduino, which is where the Pulse Width Modulation (PWM) Signals come into the Arduino. PWM pins are capable of transferring analog signals to a certain digital value. Begin the Liquid Crystal Display (LCD) with 16 characters in 2 lines. Then end the 'void setup' function with a bracket. Start the main portion of the code in a loop. Set up an array (or a string of characters) of 16 characters to put desired info on the display Set the Pulse Width Modulation value to the amount of time that the square wave is HIGH. The Arduino will calculate the amount of time the wave is HIGH, or the time interval between each wave. This changes with the Duty Value of the sig-gen. The Arduino then calculates the heart rate, then uses that number as pwm_value. .Use the 'sprintf' function to show the pwm_value (heart rate) and the message "Heart Rate: (a value)". The %3d means that the number displayed can have up to 3 digits. Set the LCD to start displaying text at cell (0,0), the uppermost-left corner. Print the pwm_value in the buffer1 message (Ex. Heart rate :79). This message was the one in the 'sprintf' function. Set a logic statement that looks for if the 'waitcounter' variable is greater than or equal to 60; this variable is originally set to 0 from the setup. You are counting UP. When it reaches 61 seconds (done with

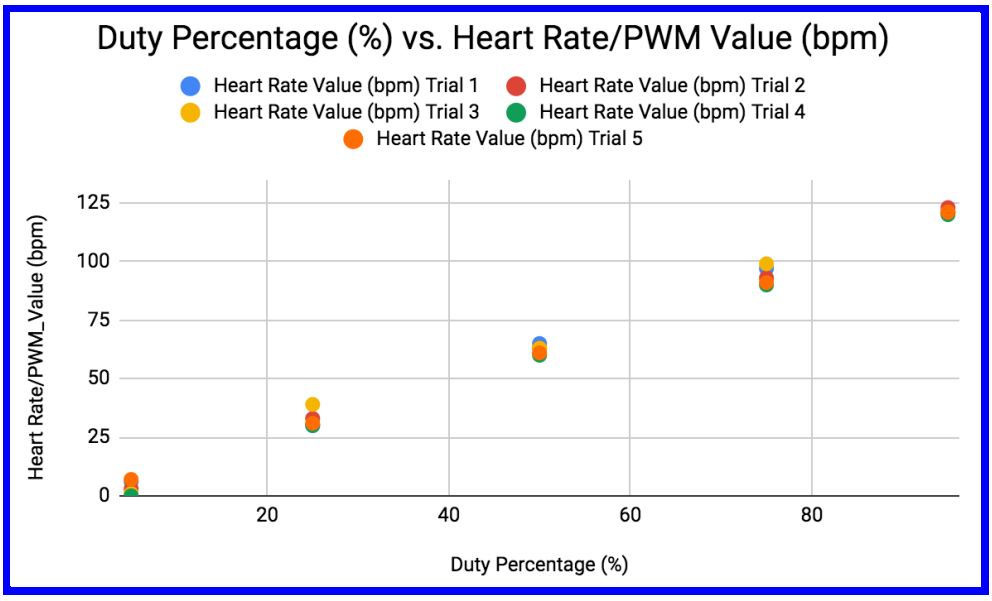
countdown), the device goes to the ELSE and does that function. Wait for a second (1000 milliseconds). Count up and every time 1 second passes, add 1 to the variable ‘waitcounter’. When it reaches 61 seconds (done with countdown), move on in the code. Once the countdown is over, start the ‘else’ function, and do the following commands. Copy the commands before, in order to continue displaying the pwm_value (heart rate) on the LCD. If the pwm_value is greater than 50 or less than 100, then move on. If the variable ‘counter’ is greater than 0 (it is until the countdown is done), then continue the next steps (the countdown). Lower the value of the variable ‘counter’ by 1. Set up an array (or a string of characters) of 16 characters to put desired info on the display. Use the ‘sprintf’ function to show the counter value (countdown) and the message “Calling: (a value) secs”. The %2d means that the number displayed can have up to 2 digits. Set the LCD to start displaying text at cell (0,1). Print the counter value in the buffer message (Ex. Calling: 20 secs). This message was the one in the ‘sprintf’ function above. Wait for 1 second (1000 milliseconds). When 1 second passes then the value of the variable ‘counter’ will be lowered by 1. Set up a new function and follow the steps in it. If the variable ‘didIcall’ is false (which it was set to in the beginning), then move on (Boolean Logic). Begin communication between the Arduino and app through the serial port. 9600 is the default communication rate. Send the phone data through the serial port. Stop sending data to the phone. Set the variable ‘didIcall’ to true, after everything is done. Format and properly end all functions to avoid compilation errors.

IV. Results

Table 1: Duty Percentage vs. Heart Rate/PWM Value (bpm)

Duty Percentage (%) vs. Heart Rate/PWM Value (bpm)						
Duty Percent (%)	Heart Rate Value (bpm) Trial 1	Heart Rate Value (bpm) Trial 2	Heart Rate Value (bpm) Trial 3	Heart Rate Value (bpm) Trial 4	Heart Rate Value (bpm) Trial 5	Average Heart Rate Value (bpm)
5.0	6.0	3.0	1.0	0.0	7.0	3.4
25.0	30.0	33.0	39.0	30.0	31.0	32.6.0
50.0	65.0	63.0	63.0	60.0	61.0	62.4
75.0	97.0	93.0	99.0	90.0	91.0	94.0
95.0	121.0	123.0	121.0	120.0	121.0	121.2

Graph 1: Duty Percentage vs. Heart Rate/PWM Value (bpm)



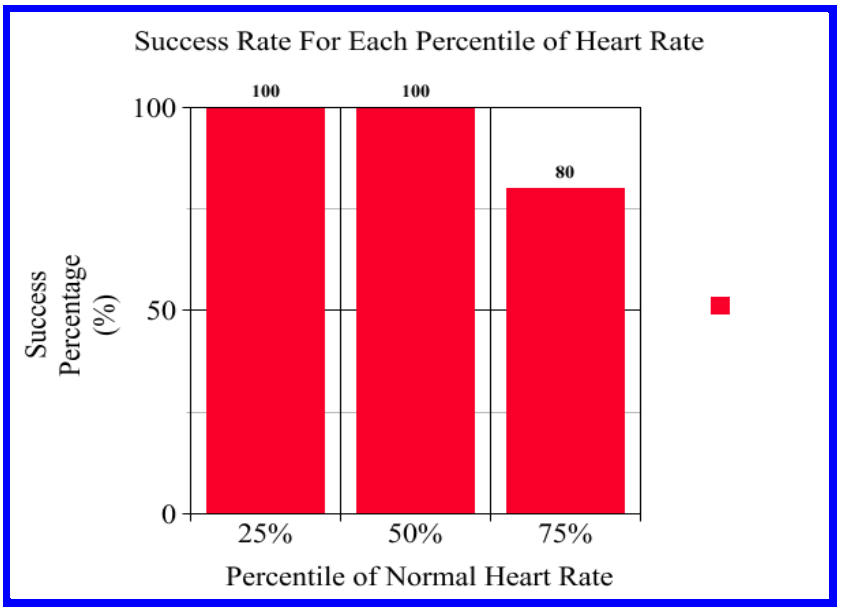
Graph 1/Table 1

In order to see how accurately the Arduino interpreted the heart rate graph from the signal generator into a number five trials were done for five different duty(time interval) percent values between waves: 5%, 25%, 50%, 75%, 95. Overall, there was a minimal margin of error and all of the data points for one specific duty value clustered around one point.

Table 2: Success Rate For Each Percentile of Heart Rate

Success Rate For Each Percentile of Heart Rate					
Percentile of Normal Heart Rate/Heart Rate Value	Trial 1 Success	Trial 2 Success	Trial 3 Success	Trial 4 Success	Trial 5 Success
25% Low (<50.0 bpm) Should Call	Yes	Yes	Yes	Yes	Yes
50% (50.0<x<100.0 bpm) Should Not Call	Yes	Yes	Yes	Yes	Yes
75% (>100.0 bpm) Should Call	Yes	Yes	Yes	Yes	No

Graph 2: Success Rate For Each Percentile of Heart Rate



Graph 2/Table 2

Five trials were conducted on the device, to make sure it worked, and demonstrated all of the set Performance Criteria.The detailed response system was trialed in three categories, a low heart rate (<50 bpm), a normal heart rate (between 50 to 100 bpm), and a high heart rate (>100 bpm).

V. Discussion

Throughout the duration of the project, many new ideas were learned in the process. After building and testing, the device met most of the performance criteria set before. The device took an numerous amount of trial and error until success, although each attempt took more knowledge acquired from research. From observations, it was seen that as one changed the duty percentage of the electronic signals the pulse generator sent out while keeping the frequency constant at 80.9 Hz, allowed for the "heart rate" that the Arduino read to change. This method was how the Arduino read the rate that the pulses were coming at (Pulse Width Modulation value) as the heart rate (by looking at the time interval between each pulse), and allowed for easy adjustments to the heart rate. Using Pulse Width Modulation (PWM) instead of analog pins, allowed for more consistent results with the values that the Arduino read as the heart rate. The PWM pins, had analog to digital converters (ADC) which allowed for the analog information form the signal generator to be transformed into digital waves that the Arduino microcontroller could understand.

Even though Arduino is an inexpensive and great way to build one's own devices there came some limitations and problems. For example, the device was a little bulky, and many ports had to be reused, and parts had to be connected creatively, because the Arduino Uno had a limited number of certain types of ports such as 5V, GND, TX, and RX. Many parts may not have received as much energy as they could have. For example the Bluetooth Module works best with 5V, but it was connected to 3.3V as the display (LCD) used up the only 5V port.

One of the main goals of the project was to make the device as cost-efficient as possible. As Arduino was used, and due to time constraints, it was difficult to get the cheapest parts on the market. Even then, the total price of the components, were about 25 dollars, which was affordable. In the future if this device were to be produced then it would be made much more smaller and cheaper to combat the limitations once presented. Making these changes, would allow the device to be available to a larger demographic and easier to use on a daily basis.

Many victims dying from heart-related diseases do not get enough medical attention in time. Victims may not be able call 911 immediately, due to the dangerous heart rate range. This project helps the victims reach the hospital quicker and potentially, the victims could be saved.

In the future, the device could become an actual watch-like device for a human. Since the signal generator acts like a human heart, in the future, a human wrist can take the place of the signal generator. The Arduino can be replaced with a smaller computer chip, with similar functions. The jumper wires will be replaced with small electronic circuits. In general, in the future, the device potentially could be much more smaller, and could be like an Apple Watch. Ideally the device would be able to measure a heart beat of a human, with a heart rate sensor similar to that of the Apple Watch and FitBit. If a person's heart rate became irregular either by speeding up or slowing down, the device would be able to detect it. Then, professionals would notified of the incident, and the person would get medical attention quickly.

The next steps, would be to add a button that when pressed, would immediately stop the countdown and immediately call 911. This would allow the victim to get help much more quickly. The device may be able to detect other heart irregularities that are specific to certain parts in the heart, such as a skipping heart rate, and Atrial Fibrillation. Users may also have the chance to customize certain features, such as age, gender, and body makeup. Also, a normal heart rate for a person can vary depending on their lifestyle. For example athletes, may have a slower resting heart rate, as the heart works more efficiently and is stronger. Based on this information, the user may have the option to enter customized heart rate ranges, depending on their lifestyle.

If the device were produced, it would likely not only be made smaller but also made much more cheaper. Due to the limitations in this project, the device may not have been as cost-efficient as it could have been. Especially bought in bulk, would allow for the prices of individual components would be much lower. The device could be available to a larger demographic than before. Specializing in the task of informing the user and getting proper medical attention when an abnormal heart rate is detected is the main goal.

VI. References:

"Your Heart Rate. What It Means, and Where on Apple Watch You'll Find It." *IPhone Service Pricing - Apple Support*, 17 Sept. 2018, support.apple.com/en-us/HT204666.

Caldwell, Serenity. "How to Check Your Heart Rate with Apple Watch." *IMore*, IMore, 2 May 2018, www.imore.com/how-check-your-heart-rate-apple-watch.

"Arduino Bluetooth Basics." *Tinkernut Labs*, 9 Mar. 2014, www.tinkernut.com/2014/03/arduino-bluetooth-basics/.

"How the Heart Works." MyHealth.Alberta.ca Government of Alberta Personal Health Portal, myhealth.alberta.ca/Health/Pages/conditions.aspx?hwid=tx4097abc.

Laskowski, M.D. Edward R. "2 Easy, Accurate Ways to Measure Your Heart Rate." Mayo Clinic, Mayo Foundation for Medical Education and Research, 29 Aug. 2018, www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979.

"When Your Heart Rhythm Isn't Normal." WebMD, WebMD, www.webmd.com/heart-disease/atrial-fibrillation/heart-disease-abnormal-heart-rhythm.

electronics+radio. "What Is a Signal Generator: Different Types." Electronics Notes, Electronics Notes,www.electronics-notes.com/articles/test-methods/signal-generators/what-is-a-signal-generator.php.

Technoblogy, www.technoblogy.com/show?20W6.

Arduino - Introduction, www.arduino.cc/en/Reference/AnalogRead?setlang=en.

Diller, Cheryl. "Create Short Pulses with a Function Generator." EDN, www.edn.com/electronics-news/4384630/Create-short-pulses-with-a-function-generator.

"Three Ways To Read A PWM Signal With Arduino." BenRipley.com, www.benripley.com/diy/arduino/three-ways-to-read-a-pwm-signal-with-arduino/.

Arduino - Introduction, www.arduino.cc/en/Tutorial/HelloWorld.

Notes, Electronics. "What Is a Signal Generator: Different Types." Electronics Notes, Electronics Notes,www.electronics-notes.com/articles/test-methods/signal-generators/what-is-a-signal-generator.php.

VII. Acknowledgements:

Ms. Murphy

Mrs. Fournier

Carla Grant (Cardiologist)