Pacemaker Project: Group 26
Mechatronics 3K04: Software Development

Hamza Alfalo (alfaloh)
Jack Coleman (colemj6)
Haaniya Ahmed (ahmedh93)
Alisa Norenberg (norenbea)
Faris Abdelqader Alterawi (abdelqaf)
Krishna Chauhan (chauhk9)

Table of Contents:

Revision History **5**
Introduction **6**
Research & Planning **7**
C++ 7
Python 8
PyQt 9
Requirements: **9**
DCM Requirements 9
Pacemaker Requirements 11
Simulink Design: **12**
Summary 12
Variables & Constants 18
Design Considerations 21
Input Subsystem 21
Serial Communications - Receive 23
Serial Communications - Transmit 25
Mode Decision 28
AOO 29
VOO 30
AAI 31
VVI 31
AOOR 32
VOOR 32
AAIR 33
VVIR 33
Rate Adaptive Subsystem 34
Output Subsystem 37
DCM Design: **38**
Design Considerations 38
Class Diagram 47
Classes 48
Methods 49
Testing Results - DCM **62**
Registration - Case 1: 62
Registration - Case 2: 62

Registration - Case 3:.....63
Registration - Case 4:.....63
Registration - Case 5:.....64
Login - Case 1:.....65
Login - Case 2:.....65
Login - Case 3:.....66
Login - Case 4:.....66
Main Page Test Cases:.....67
Sign Out Button:.....67
Checking Saved Parameters When Switching Modes:.....67
Checking Saved Parameters When signing out and signing back in:.....69
Checking Reset to Nominal Button:.....70
Checking Change Ui Button:.....72
Checking Device Status:.....75
Checking Send to Pacemaker Button/Communication Status:.....76
Checking Graph:.....77
Problems and Challenges:.....84
DCM.....84
Assignment 1.....84
Assignment 2.....84
Simulink.....85
Assignment 1.....85
Assignment 2.....85
Testing Results - Pacemaker.....86
AOO - Case 1.....86
AOO - Case 2.....86
VOO - Case 1.....87
VOO - Case 2.....87
AAI - Case 1.....88
AAI - Case 2.....88
AAI - Case 3.....89
VVI - Case 1.....89
VVI - Case 2.....90
VVI - Case 3.....90
Additional Tests.....91
Rate Adaptive Testing.....93
Assurance Cases.....97
Future Project Changes.....101
DCM.....101

Assignment 1.....101
Assignment 2.....102
Simulink.....102
Assignment 1.....102
Assignment 2.....103
References:..... 104

Revision History

Revision	Description	Date (DD/MM/YYYY)	Contributor(s)
Assignment #1	Pacemaker design (AOO, VOO, AAI, VVI), DCM development, documentation write-up	25/10/2024	Haaniya Ahmed, Alisa Norenberg, Hamza Alfalo, Krishna Chauhan, Jack Coleman, Fares Al Terawi
Assignment #2	Added AOOR, VOOR, AIIR, VIIR modes. Serial communications with DCM and pacemaker. Rate adaptive pacing.	29/11/2024	Haaniya Ahmed, Alisa Norenberg, Hamza Alfalo, Krishna Chauhan, Jack Coleman, Fares Al Terawi

Introduction

Pacemakers are vital devices for people with abnormal heart conditions, allowing them to experience a natural pulse through electrical stimulation. This project focuses on developing a model of a pacemaker, simulating its functionality across different pacing modes to treat different cardiac conditions.

The primary reason to develop this model is to demonstrate how a pacemaker operates in multiple modes, each tailored to specific cardiac conditions. Some modes provide continuous pacing to ensure the heart maintains an appropriate rhythm, while others are more adaptive, only activating when the heart's natural pacemaking function falls short. The simulation will demonstrate how these different modes function and respond to irregular heartbeats, ensuring that each mode effectively supports the heart when needed. Simulating these modes allows a deeper understanding of the underlying mechanics of pacemaker functionality and how these devices are designed to improve patient outcomes in clinical settings.

There are two parts to this project–a model and a Device Controller Monitor (DCM). The model was developed on MATLAB Simulink. Simulink, a powerful simulation environment, enabled the building, simulation, and refinement of the pacemaker's design by creating detailed models of its behaviour and logic. This was used in parallel to Heartview, a specialized cardiac simulation tool, to simulate the heart's activity and validate the design's functionality.

The DCM serves as an important tool for healthcare professionals, allowing them to observe and adjust the operational parameters of medical devices in real-time. This ensures that each device can respond according to the specific needs of individual patients. The DCM is also designed to support a comprehensive data management system that can accommodate multiple users. By enabling clinicians to customize device settings based on each patient's unique health profile, the DCM enhances the precision of patient care. This feature not only reflects the device’s flexibility but also its potential to improve patient outcomes through tailored interventions.

In conclusion, by developing this project, it highlights the importance of cardiac health technology, safety in medical devices, and the process of software development. The proceeding sections will detail the research and development methods for both the DCM and Simulink model.

Research & Planning

Before starting this project, a GitHub repository was created to allow proper and efficient collaboration between team members. Teams were then split into two subteams, one DCM and one Simulink. Each team read the required documentation to ensure a proper understanding of the project. Two hardware kits (each kit containing a pacemaker and a heart simulation) were also obtained (**Figure 1**).

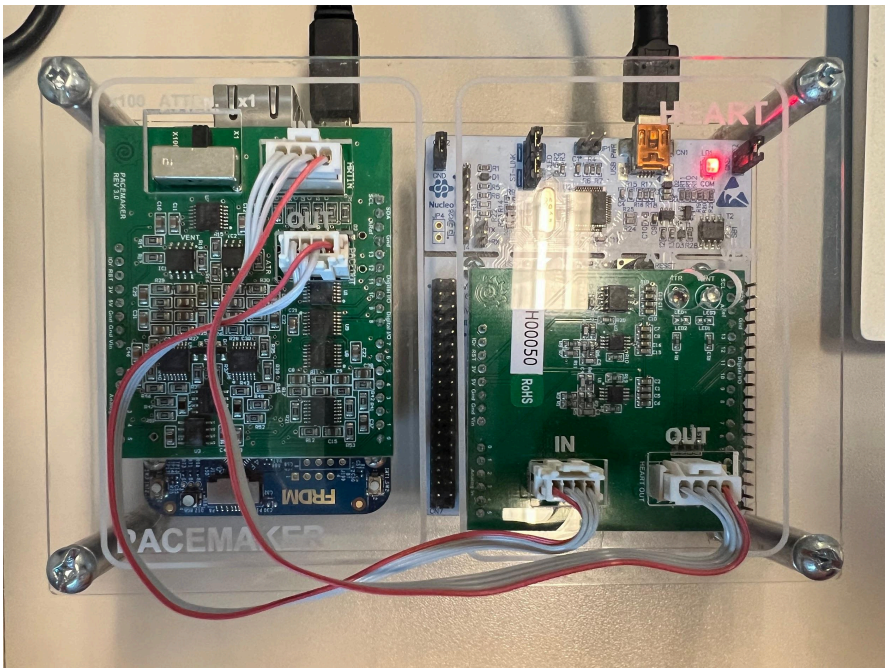

Figure 1. Image displaying the hardware kit showcasing the pacemaker (left) and the heart (right)

For the pacemaker, all the pacing modes were created in their own MATLAB Simulink, and then were combined in the end; this was done to ensure efficient work among subteam members. Separate files also allowed for easier testing of each mode developed.

The first step in developing the DCM was to determine the programming language and GUI framework for the project. Based on the team's qualifications, the options included C++ and Python.

C++

In C++, Qt and GTK were the main options for building GUIs. Qt offers a comprehensive set of tools for creating detailed and customizable user interfaces. It supports complex layouts, and widgets, and integrates well with C++ for efficient performance. This makes it very convenient for applications where fine control over the UI is crucial, like in medical devices or systems requiring real-time updates. Developing GUIs in C++ can be extremely time-consuming. Every aspect must be managed from widget creation and event handling to memory management, manually. GTK is another option, a bit simpler and

more lightweight than Qt, but it's still flexible enough for most projects, though it lacks some of the advanced features Qt provides.

For data storage, C++ provides developers with absolute control over file management. If local storage is chosen, all handling, file reading, writing, and error-checking are done manually. This allows developers to optimize everything but adds complexity. C++ can also integrate with databases like SQLite or MySQL, which provide structured data management. Databases are great for larger, scalable systems, they require more setup and management in C++, including establishing database connections and managing resources manually. This can get relatively slow compared to local storage, especially on smaller projects.

When it comes to serial communication, C++ is useful for low-level hardware control and live communication since speed and precision are of the utmost importance. However, handling serial ports in C++ requires setting up data buffers, and protocols, which requires more time and resources. This approach gave developers the highest level of control but at the cost of simplicity and development time [5].

Python

In Python, two main options for GUIs were determined: PyQt and Tkinter. PyQt is the Python binding for the Qt framework, it offers a reliable way to build modern, dynamic interfaces. It simplifies lat management, event handling, and widget creation, allowing developers to focus on functionality without worrying about many underlying complexities. PyQt's flexibility makes it great for more complex applications that require interactive UIs. Tkinter, however, is much more basic. It's easier to use for simple interfaces, but it lacks the customization options of PyQt, making it less suitable for applications that need more intricate designs or features.

For data management, Python is known for making things easy. If local storage is used, Python's built-in file handling lets developers read and write files in just a few lines. It's perfect for smaller projects where just storing and retrieving basic user information or settings. If more structured data handling is needed, Python works seamlessly with databases like SQLite [5]. SQLite is a lightweight database that's part of Python's standard library, making it easy to set up and manage. Python's database libraries take away a lot of complexities related to data management, allowing users to implement and handle structured data without dealing with low-level details.

For serial communication, Python's pySerial is a high-level library that simplifies the entire process. It can set up a serial connection, send, and receive data with just a few lines of code. It doesn't offer the same level of control as C++, for most applications where rapid development and ease of use are important, Python is preferable. Python takes away the hassle of manually managing buffers and protocols, making it the better option when performance isn't the highest priority [5].

PyQt

Python and PyQt were chosen for this project because the focus is on rapid prototyping rather than scalability. Given that this software is intended to support up to 10 patients under the guidance of a healthcare practitioner, Python offers the simplicity and speed required to meet these needs without over-complicating the development process [6].

With minimal backend management requirements and a smaller user base, there's no immediate need for a database. Instead, locally storing data is both simpler and safer for this stage of development, as it reduces complexity and minimizes security risks associated with data storage. PyQt was selected specifically for its flexibility, providing an intuitive way to build and customize the user interface efficiently, while still allowing the system to remain adaptable if future changes are needed.

Requirements:

DCM Requirements

The DCM's login page must allow users to enter a username and password and provide a button to submit these credentials to initiate the login process. The system must validate the existence of the username and verify that the password entered matches the stored password associated with that username. If the login attempt fails due to an incorrect username or password, an error message must be displayed to inform the user of the issue. The login page must also include a way to access the registration.

The registration functionality must enable new users to create an account by entering a username, and password, which are then stored. The system must verify that the chosen username is not already chosen. If the registration attempt fails due to a taken username, an error message must be displayed. The registration function must also provide a means to return to the login functionality.

The system is limited to a maximum of 10 user accounts. Once this limit is reached, new user registrations must be disabled, and a message displayed to indicate that no additional accounts can be created.

Upon successful login, users must access a main page where they can select a pacing mode from AOO, VOO, VVI, AAI, AOOR, VOOR, VVIR, and AAIR. The main page must allow users to adjust mode-specific parameters within the predefined bounds. Users must be able to save these settings, which should persist between sessions, ensuring that settings remain available after logging out and back in. A logout function must also be provided to allow users to exit their session and return to the login page.

The user must have access to the pacemaker connection status and communication status. Correct parameters must be sent to the pacemaker, and the egram data must be received from the pacemaker and subsequently graphed. All communication must be done via UART.

Table 1: Required parameters grouped by pacing mode

Parameter	AOO	AAI	VOO	VVI	AOOR	AAIR	VOOR	VVIR
Lower Rate Limit	X	X	X	X	X	X	X	X
Upper Rate Limit	X	X	X	X	X	X	X	X
Maximum Sensor Rate					X	X	X	X
Atrial Amplitude	X	X			X	X		
Ventricular Amplitude			X	X			X	X
Atrial Pulse Width	X	X			X	X		
Ventricular Pulse Width			X	X			X	X
Atrial Sensitivity		X				X		
Ventricular Sensitivity				X				X
VRP				X				X
ARP		X				X		
PVARP		X				X		
Hysteresis		X		X		X		X
Rate Smoothing		X		X		X		X
Activity Threshold					X	X	X	X
Reaction					X	X	X	X

Time								
Response Factor					X	X	X	X
Recovery Time					X	X	X	X

Pacemaker Requirements

Using Simulink state flows, the following pacing modes must be implemented for a pacemaker in a permanent state:

AOO: Atrial pacing, no chambers sensed, no response to sensing
AAI: Atrial pacing, atrium sensed, pacing inhibited in response to sensing
VOO: Ventricular pacing, no chambers sensed, no response to sensing
VVI: Ventricular pacing, ventricle sensed, pacing inhibited in response to sensing
AOOR: Atrial pacing, no chambers sensed, no response to sensing, rate modulation
AAIR: Atrial pacing, atrium sensed, pacing inhibited in response to sensing, rate modulation
VOOR: Ventricular pacing, no chambers sensed, no response to sensing, rate modulation
VVIR: Ventricular pacing, ventricle sensed, pacing inhibited in response to sensing, rate modulation

The Simulink model must use the programmable parameters required for each pacing mode. The parameters required for Assignment 1 involve the pulse characteristics (width and amplitude), rate characteristics (limits and delays), as well as which chambers are being paced. They are listed in a table below according to mode:

Table 2: Required parameters grouped by pacing mode

Parameter	AOO	AAI	VOO	VVI	AOOR	AAIR	VOOR	VVIR
Lower Rate Limit	X	X	X	X	X	X	X	X
Upper Rate Limit	X	X	X	X	X	X	X	X
Maximum Sensor Rate					X	X	X	X
Atrial Amplitude	X	X			X	X		
Ventricular			X	X			X	X

Amplitude								
Atrial Pulse Width	X	X			X	X		
Ventricular Pulse Width			X	X			X	X
Atrial Sensitivity		X				X		
Ventricular Sensitivity				X				X
VRP				X				X
ARP		X				X		
Activity Threshold					X	X	X	X
Reaction Time					X	X	X	X
Response Factor					X	X	X	X
Recovery Time					X	X	X	X

Hardware hiding must be used for the model to allow for the pin mapping to be changed without affecting the rest of the model. This also allows for the model to be abstracted away from the hardware, which will make the model easier to modify as its required behaviour becomes more complex.

Simulink Design:

Summary

This section will discuss the use of Simulink in conjunction with the FRDM-K64F hardware board to simulate the pacemaker's functionality. The Simulink model was developed to replicate the various pacing modes along with their respective sensing and timing mechanisms. By integrating the hardware with the simulation environment, the pacemaker's response to different cardiac conditions was tested and validated in real time.

Figure 2 describes the different pacing modes available in the pacemaker. The four primary modes as discussed are AOO, VOO, AAI, and VVI. In AOO, the pacemaker delivers atrial pacing at regular intervals without sensing any heart activity, meaning there is no detection of natural heart signals, and the pacing output is directed to a designated GPIO pin for atrial pacing. Similarly, in VOO mode, the pacemaker provides ventricular pacing at regular intervals without sensing ventricular activity, and the output is sent to a GPIO pin for ventricular pacing.

In contrast, AAI mode introduces sensing functionality. It delivers atrial pacing but inhibits the pacing if intrinsic atrial activity is detected. The system senses atrial signals via a specified GPIO pin and sends a pacing output if needed. Finally, in VVI mode, the pacemaker offers ventricular pacing, but it is inhibited if intrinsic ventricular activity is detected. The system senses ventricular activity using a specific GPIO pin for ventricular input and outputs pacing signals only when no natural ventricular activity is present. Each mode is designed to address specific cardiac pacing needs, either continuously pacing or responding to the detection of natural heart signals to regulate heart function effectively.

The last thing that should be observed in the testing is the lack of inhibition of heartbeats below the lower rate limit. Given that the lower rate limit is set to an average heart rate (60 ppm), anything below is due to a biological issue [1]. Thus, pacing is required.

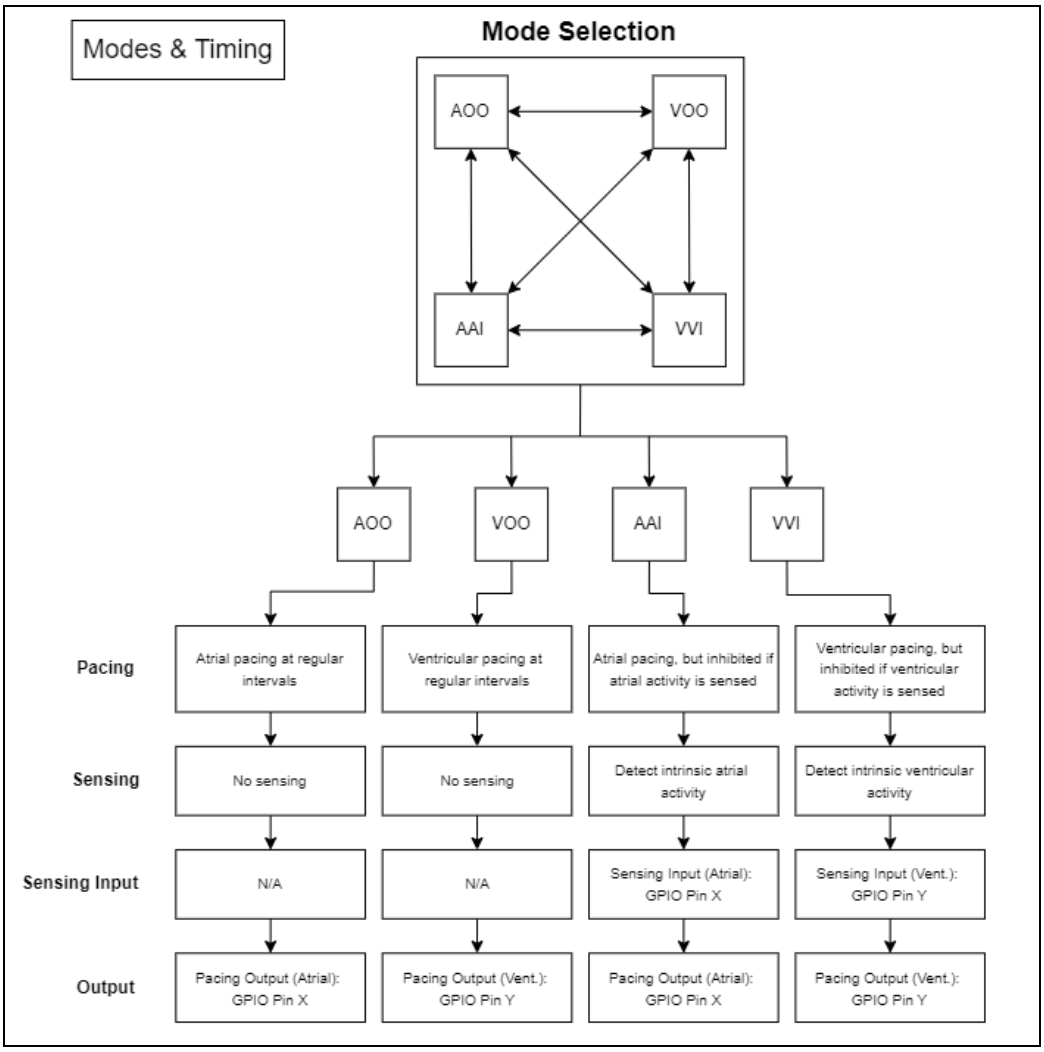

Figure 2: Mode selection logic for each of the four modes - AOO, VOO, AAI, and VVI,

The following flowchart (**Figure 3**) illustrates the general operation of a pacemaker, specifically for the VVI and AAI modes, which actively monitor and regulate the heart's electrical activity. The process begins with the pacemaker becoming active and entering a sensing mode continuously monitoring the heart's signals. If the detected signal is equal to or greater than the sensitivity threshold, the pacemaker assumes it is a natural heartbeat and takes no action. However, if the signal is below this threshold, the pacemaker generates a pulse using PWM and sends this electrical signal to the heart to initiate a pulse.

After either a natural or paced heartbeat, the system starts a timer to check whether the pacemaker is in a refractory period, a brief interval during which the pacemaker ignores any additional signals to prevent unnecessary or premature pacing. If the refractory period is active, incoming signals are ignored. Once the refractory period ends, the pacemaker resumes normal sensing and continues the cycle. This process ensures that the heart maintains a regular rhythm through constant monitoring and pacing when necessary.

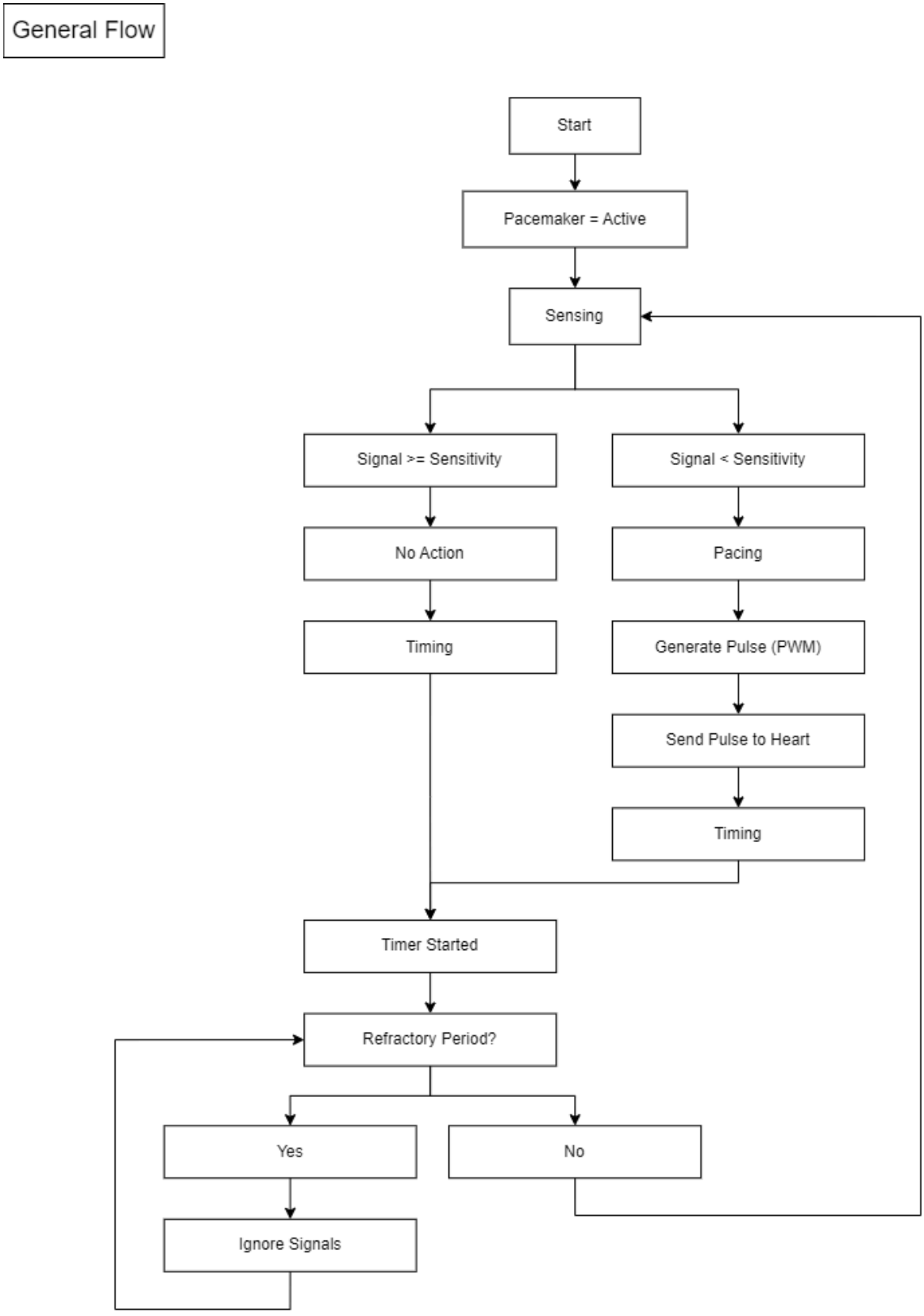

Figure 3: Flow of logic for sensing modes: AAI, VVI, AAIR, VVIR.

The other modes that were included were AOOR, VOOR, AAIR, and VVIR. They follow the same logic as their non-rate adaptive counterparts. However, the differences are discussed in Table #, and can be seen in the Rate Adaptive Test Cases. The main comparison lies within the fact that the rate adaptive modes are tied to the on-board accelerometer, and whether the board is being shaken or not.

Table 3: Differences in rate adaptive and non-rate adaptive modes

Board Status	**AOO/VOO**	**AOOR/VOOR**	**AAI/VVI**	**AAIR/VVIR**
At Rest	Pacing regardless of whether there is a natural heartbeat or not.	Pacing regardless of whether there is a natural heartbeat or not.	Pacing when there is no natural heartbeat. Also pacing when heartbeat is less than lower rate limit.	Pacing when there is no natural heartbeat. Also pacing when heartbeat is less than lower rate limit.
Shaken	No change.	Pacing gets closer (more frequent).	No change.	If heartbeat is greater than lower rate limit, pacing begins.

The following table summarizes the pins used in this project, specifying each pin's input or output designation, along with the assigned function. This will be further discussed in the proceeding sections.

Table 4: Summary of pins used for FRDM-K64F board.

Pin	**Input/Output**	**Function**
ATR_CMP_DETECT (D0)	Input	Used in atrium sensing, outputs 1 when the atrial signal voltage exceeds the threshold voltage, and outputs 0 when below the threshold.
VENT_CMP_DETECT (D1)	Input	Similar function as D0, except for ventricular sensing.
PACE_CHARGE_CTRL (D2)	Output	Controls charging of primary capacitor (C22). When set to 1, PWM charges C22; when 0, PWM is disconnected.
VENT_CMP_REF_PWM (D3)	Output	Charges a capacitor via PWM to maintain a threshold voltage for ventricular heartbeat sensing. The threshold is linearly proportional to the PWM duty cycle.
Z_ATR_CTRL (D4)	Output	Connects the impedance circuit to the atrial

		ring electrode to evaluate the impedance between the atrial electrodes and the heart tissue.
PACING_REF_PWM (D5)	Output	Charges the pacing circuit's primary capacitor (C22).
ATR_CMP_REF_PWM (D6)	Output	Operates identically to D3, but is used for atrial heartbeat sensing.
Z_VENT_CTRL (D7)	Output	Functions the same as D4 but for the ventricle, connecting the impedance circuit to the ventricular ring electrode.
ATR_PACE_CTRL (D8)	Output	Discharges primary capacitor through atrium by setting pin to 1. Current will flow if 1; if 0, no current flows.
VENT_PACE_CTRL (D9)	Output	Operates identically to D8, but for the ventricle.
PACE_GND_CTRL (D10)	Output	Allows current to flow between the ring and tip electrodes in the atrium/ventricle. Pin must be 1 to enable charge flow through the switch into the blocking capacitor (C21).
ATR_GND_CTRL (D11)	Output	Allows for the discharge of the blocking capacitor through the atrium, preventing charge buildup.
VENT_GND_CTRL (D12)	Output	Operates identically to D11, but for the ventricle.
FRONTEND_CTRL (D13)	Input	Controls the activation of the sensing circuity. When 1, the circuitry outputs heart signals. When 0, the circuitry is disconnected from the patient, outputting no signal.
LED_GREEN	Output	Used as the indicator of Sensing When 1, turns on indicating sensing is occurring When 0, turns off
LED_BLUE	Output	Used as the indicator of pacing When 1, turns on indicating pacing is occurring When 0, turns off
LED_RED	Output	Used to see if the accelerometer has turned on and detecting speeds.

VENT_SIGNAL (A1)	Output	A signal which represents what is happening in the ventricle real-time.
ATR_SIGNAL (A0)	Output	A signal which represents what is happening in the atrium real-time.

Variables & Constants

The variable definitions and values used in the input subsystem are described in **Table 5**.

Table 5: Input variable descriptions and ranges in the Simulink Input subsystem

Variable	Description	Value/Range
ATR_CMP_DETECT_D0	Used in atrium sensing through the digital read block.	● 1 - the atrial signal voltage exceeds the threshold voltage. ● 0 - when below the threshold.
VENT_CMP_DETECT_D1	Used in ventricular sensing through the digital read block.	Same as D0, but in terms of the ventricle signal.
FRONTEND_D13	Activates the sensing circuitry through the digital read block.	● 1 - the circuitry outputs heart signals. ● 0 - the circuitry is disconnected from the patient, outputting no signal.
Atrium_Pace_Width	Atrium pace width, which is the pulse width sent to the heart.	● Default model value: 5 ms ● Allowable range: 1-11 ms (HeartView)
Atrium_Amplitude	Atrium amplitude is the amplitude of the paced signal in the atria.	● Default model value: 5 Volts ● Allowable range: 0.5-5 Volts
Atrium_RP	Atrium refractory period, which is the time in between each discharged pulse.	● Default model value: 320 ms ● Allowable range: 150-500 ms
Atrium_Ref_PWM	Atrium refractory PWM (i.e., the duty cycle). This value is determined in the **Input Subsystem** section.	51 (NOT user adjustable, was determined experimentally)
Lower_Rate_Limit	Lower heart rate limit, which defines the longest allowable	Is user adjusted, where the value will range based on different populations

	pacing interval.	(based on factors including age) [2] ● Default model value: 60 ppm ● Allowable range: 30-175 ppm
Upper_Rate_Limit	The upper heart rate limit is the minimum time between a ventricular event and the next pace.	Is user adjusted, where the value will range based on different populations (based on factors including age) [2] ● Default model value: 120 ppm ● Allowable range: 50-175 ppm
MSR	Maximum sensor rate. This is compared with the upper rate limit, where the minimum value is used for the MSR.	Is user adjusted, where the value will range based on different populations (based on factors including age) [2] ● Allowable range: 50-175 ppm
Ventricle_PW	Ventricle pulse width, which is the pulse width sent to the heart.	● Default model value: 5 ms ● Allowable range: 1-11 ms (HeartView)
Ventricle_Amplitude	Ventricle amplitude is the amplitude of the paced signal in the ventricles.	● Default model value: 5 Volts ● Allowable range: 0.5-5 Volts
Ventricle_RP	Ventricle refractory period, which is the time in between each discharged pulse	● Default model value: 320 ms ● Allowable range: 150-500 ms
Ventricle_Ref_PWM	Ventricle refractory PWM (i.e., the duty cycle). This value is determined in the **Input Subsystem** section.	53 (NOT user adjustable, was determined experimentally)
MODE	An input parameter that can be set between 1-8 to activate one of the eight modes.	● 1 - AOO ● 2 - VOO ● 3 - VVI ● 4 - AAI ● 5 - AOOR ● 6 - VOOR ● 7 - AAIR ● 8 - VVIR
Activity_Threshold	A variable that is compared to the activity level (from the accelerometer signal) to determine a value to add to the previous target rate.	● Allowable range: V-Low, Low, Med-Low, Med, Med-High, High, V-High
Response_Factor	The response factor determines	● Allowable range: 1-16

	the incrementation to increase or decrease the desired pace rate.	
Reaction_Time	The reaction time is the time required to increase the pace rate from the lower rate limit to the upper rate limit.	• Allowable range: 10-50 s
Recovery_Time	The recovery time is the time required to decrease the pace rate from the maximum sensor rate to the lower rate limit.	• Allowable range: 2-16 min

The upper and lower rate limits are parameters (among others) that will likely be determined by a physician. In general, the upper and lower rate limits will vary between multiple populations based on factors such as age [2], as depicted in **Figure 4**. Therefore, 2 additional test cases were completed with a different lower rate limit to demonstrate product functionality for various populations.

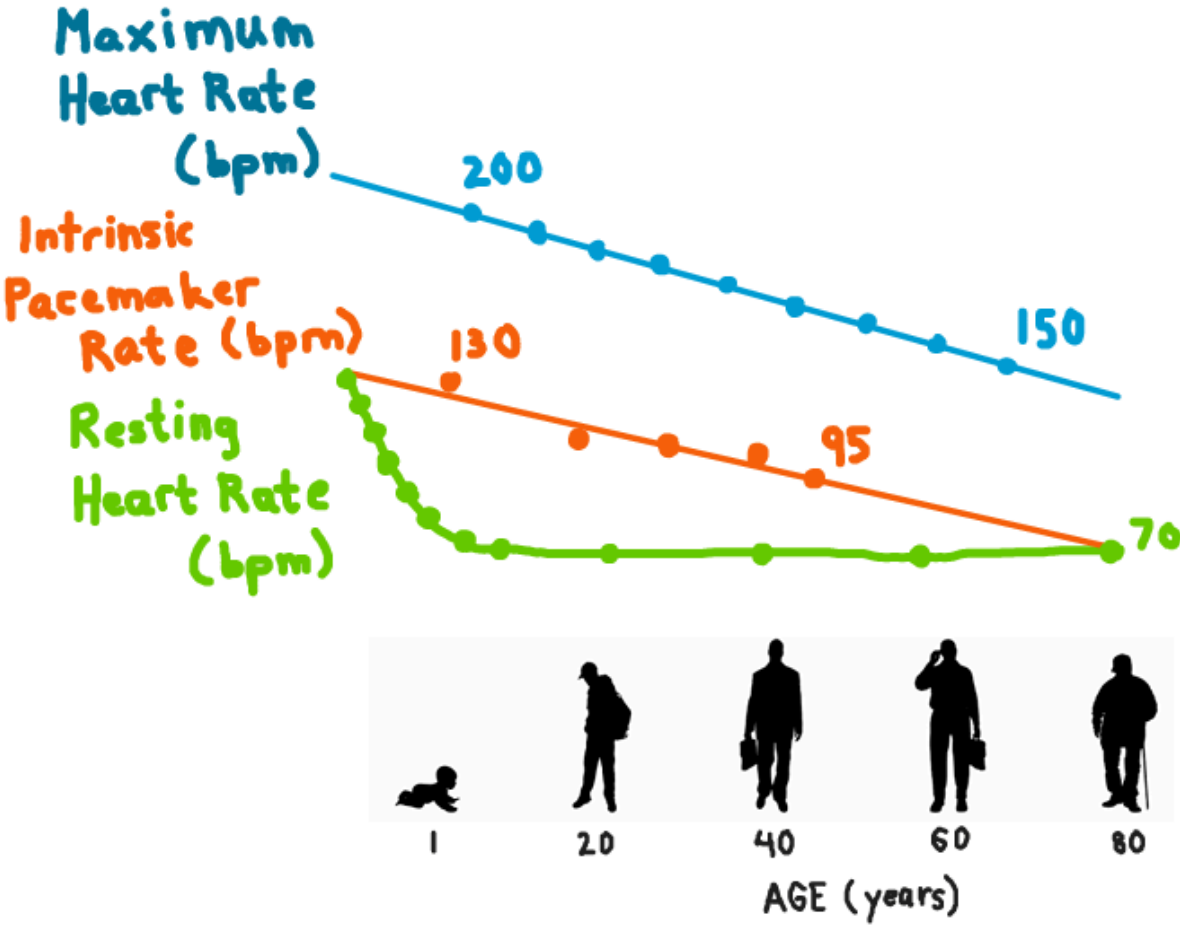

Figure 4: Average maximum and resting heart rate as age increases, adapted from [2].

Design Considerations

The design considerations for this pacemaker simulation focused on functionality, safety, and user control. Each of the eight pacing modes was initially designed in separate files to ensure that each mode was correctly implemented and tested before being integrated into a single system. This modular approach allowed for thorough testing of individual modes and helped prevent conflicts during the integration process.

Mode switching was also incorporated to allow users to switch between the eight modes by selecting a corresponding value (1-8). There is no option for a mode outside the range, since it is selected via a button on the DCM.

Safety was also considered in the rate adaptive MATLAB function block In the Rate Adaptive Subsystem. Once a preliminary desired pace rate is calculated, it enters an if loop that ensures that the desired pace rate is within the bounds of the lower rate limit and the maximum sensor rate. This way, the desired pace rate will always be within the allowable range and the user will not be harmed by pacing that is too low or too high.

When incorporating the rate adaptive modes, a red LED was incorporated into the design, this red LED would start flashing once an acceleration was detected by the sensor, this design feature also came pretty handy when it came to debugging.

The LEDs are a way to account for the variety of patients, some who may need visual indicators to explain what is going on. As well, it can provide real-time visual feedback. For healthcare providers, it is a quick and easy diagnostic method, allowing them to assess the pacemaker without relying on understanding the technical aspects of the device or complex equipment.

Input Subsystem

The input subsystem was designed to manage programmable parameters defined in **Table 1**, which will later be connected to the DCM for user-based adjustments, as shown in **Figure 5**. These adjustments will modify the behaviour of the pacemaker such as its pulse rate, pulse width, pulse amplitude, sensitivity in the sensing circuit, pacing modes, refractory period, etc. The atria and ventricles have independent programmable variables, based on varying requirements for each set of chambers. Furthermore, by implementing an input subsystem, it acts as a form of hardware hiding because it adds a layer of abstraction where hardware-specific details are separated from the core model logic. The different modes can be selected via the DCM. This will be further expanded in the Mode Decision and Serial Communications section.

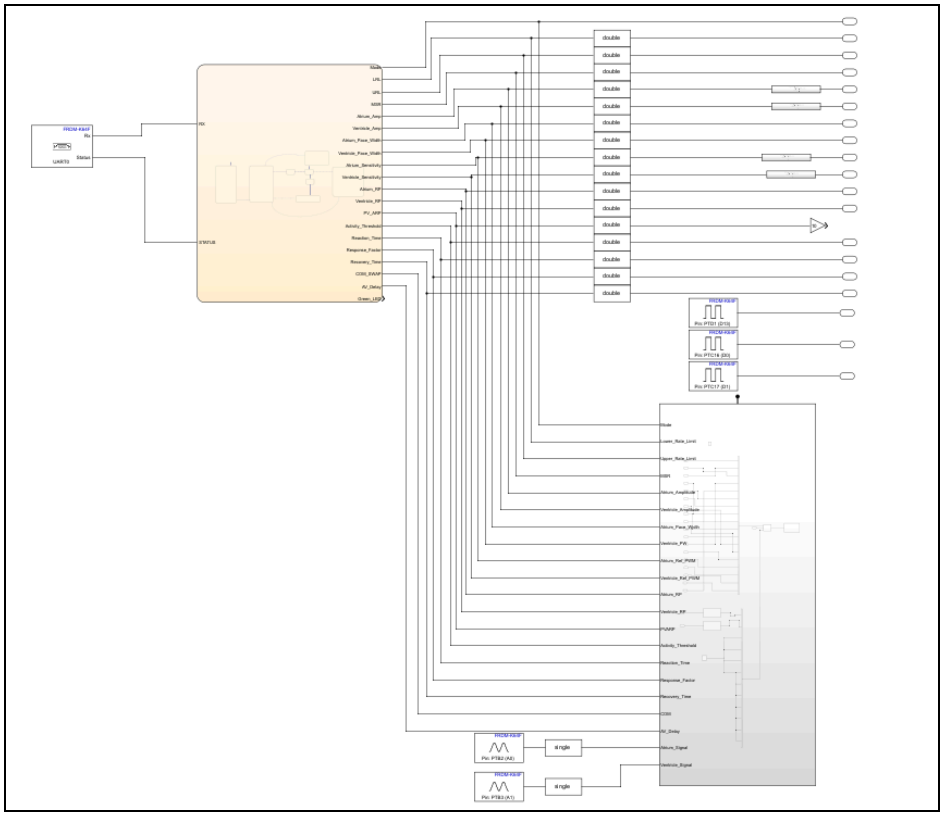

Figure 5: Input parameter subsystem that will be linked to the DCM for user adjustments.

One parameter that will not be user-adjusted in the input subsystem is the PWM duty cycle, denoted as the atrium or ventricle PWM in the model. The PWM duty cycle is linearly proportional to the capacitor voltage and it was set to 70%, making the average voltage 3.5 V, as shown below:

$$PWM\ duty\ cycle\ =\ 53/100 = 53\%$$

$$Average\ Voltage\ =\ 5\ V\ \times\ 0.53\ =\ 2.65\ V\ \text{ to the heart}$$

Safety is also being considered by using a duty cycle of less than 100% since high voltages are not used unnecessarily in the pacemaker. Additionally, a 70% duty cycle provides a longer pulse duration capture [3], as shown in **Figure 6**. This allows the detection of very slow signals under 60 bpm, a characteristic and symptom of bradycardia arrhythmia (a conductive disorder) [4].

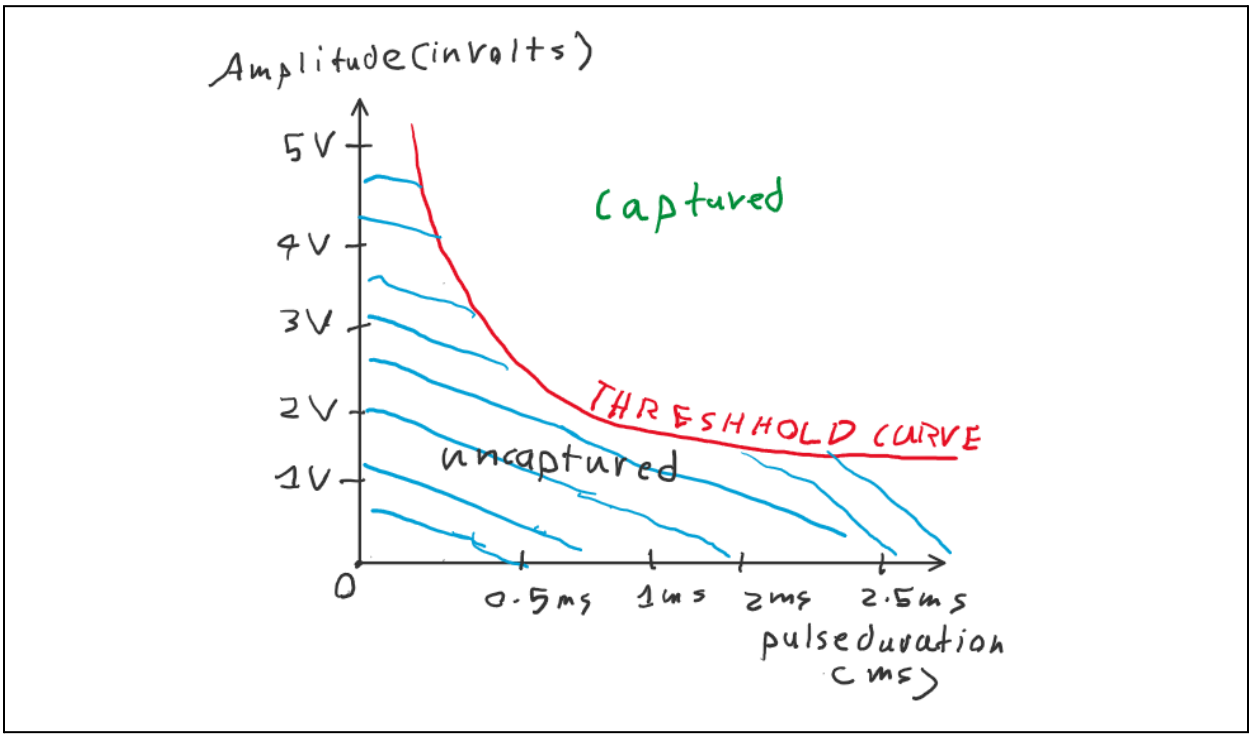

Figure 6: Estimation of the strength-duration curve showing the relationship between voltage and pulse width capture

However, when testing multiple cases of low natural heartbeats below 50 bpm on the HeartView software, the AAI, VVI, AAIR, and VVIR modes would not work as expected. Specifically, the pacemaker would be inhibited even with a natural heartbeat below the lower range limit and above the upper range limit. This inhibitory behavior should only be observed for paces within the lower and upper rate limits. Therefore, the PWM value was lowered to 51 for the atrium and 53 for the ventricle to increase the sensing pulse duration. This change not only further lowers the average 3.5 voltage, but also allows the AAI, VVI, AAIR, and VVIR modes to work as expected. Specifically, the pacemaker was no longer inhibited in heartbeats beyond both ends of the bounds. This will be demonstrated in the Testing Results section.

Serial Communications - Receive

The serial communication between the DCM and the Simulink was done using the serial receive block, the serial receive block sends two outputs; STATUS and RX. The RX output was decided to be a 20 element array containing all the necessary inputs for the Simulink to operate, this array is sent by the DCM to allow for the changing of values from the DCM interface.

The Status value just ensures that there is a connection, it sets itself to 0 when there is a successful connection. The Simulink chart will not operate unless this value is set to 0 (**Figure 7**).

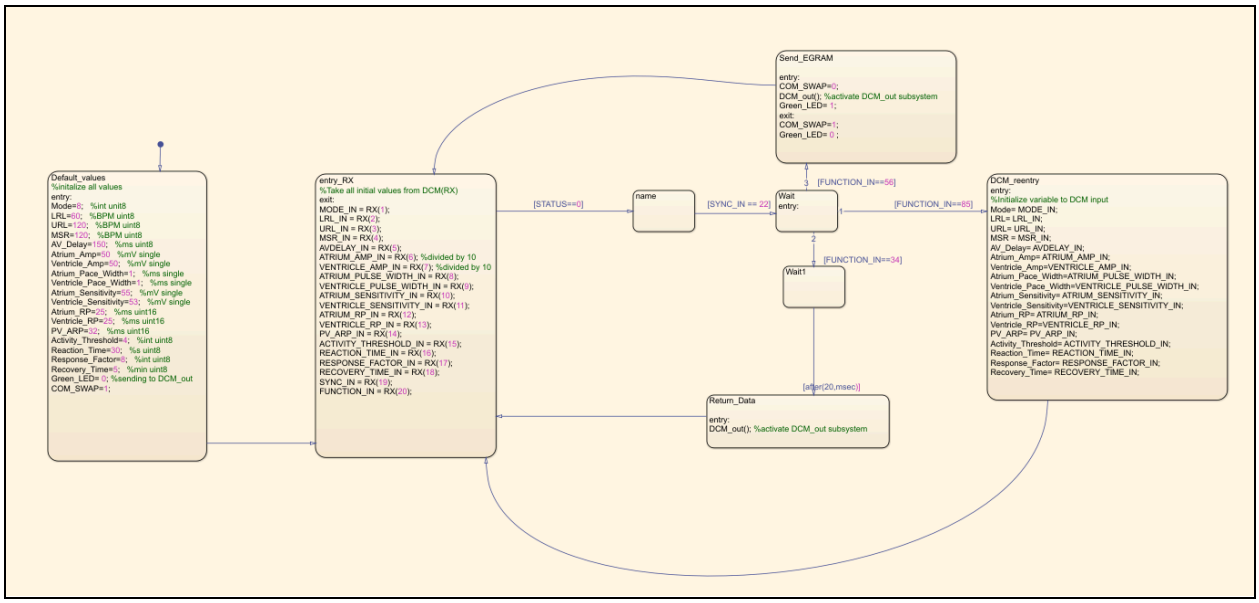

Figure 7: Chart displaying serial communication between Simulink and DCM

The chart starts off with setting initial values for all the parameters, it then updates those values to the values sent by the DCM. After, there are 3 possible paths that the chart can take, these paths depend on the setting of Function_IN that the DCM sends (Table 6). Having 3 different functions settings enhances the safety of the Pacemaker, since it insures that no data is accidentally received or sent.

Table 6: Setting for FUNC for serial communications

Function Setting	Hexadecimal	Function
34	0x22	Sets COM = 0, and sends back mode settings then sets COM back to 1
56	0x38	Send back egram data to be displayed on DCM
85	0x55	Updates mode settings from DCM to Simulink again

Another important note is that the chart will also not operate unless the SYNC_IN value is set to 22 (hexadecimal 0x16). This is done to ensure that a proper connection is always there, since the DCM looks for this hexadecimal only, any noise or extra signals coming from the cable that the connection is being made with is removed.

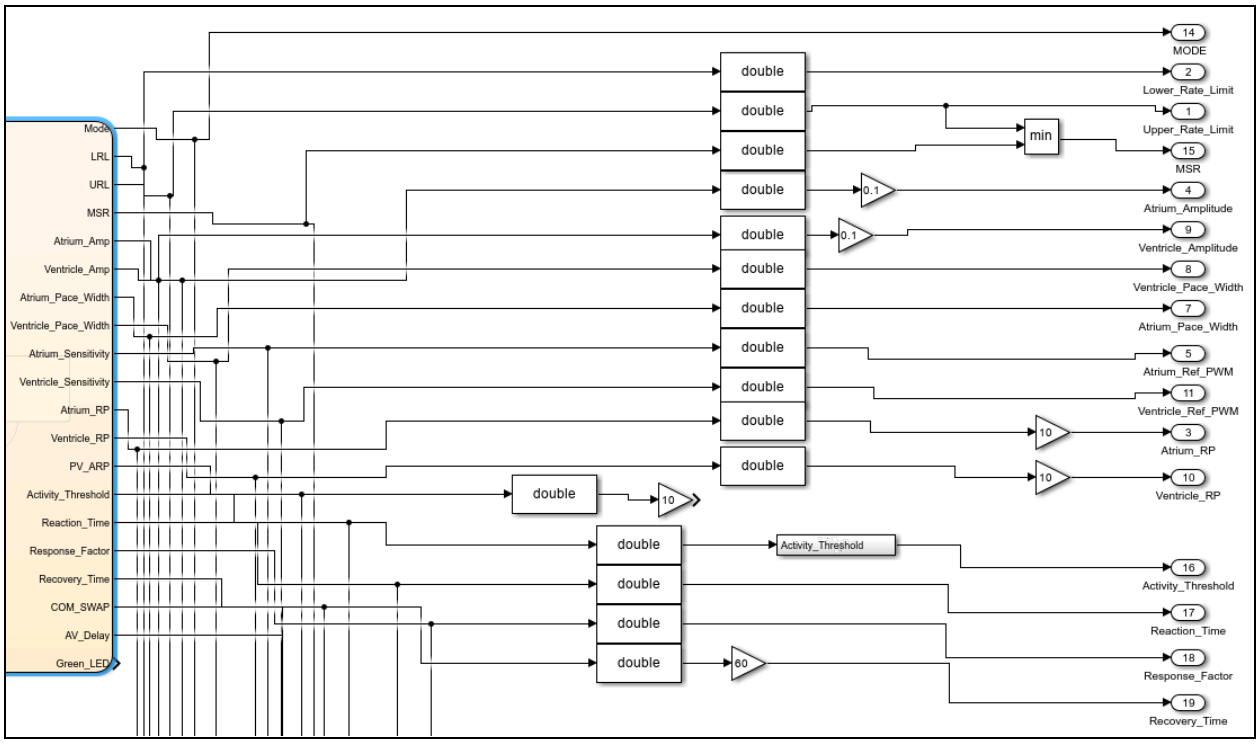

Figure 8: Calculations and castings done to the input subsystem

Since the DCM can not send decimals or values larger than 256 (since uint8 has a range of 0-256) certain modifications were performed on these values when sent to the Simulink. The Simulink then reverses back those modifications using the gain blocks in order to finally receive the actual correct values. This can be observed in the 0.1 gain on Atrium and Ventricle amplitude and the 10 gain on Atrium and Ventricle RP (**Figure 8**). A gain of 60 was also added to recovery time to convert it form minutes to seconds. For consistency reasons, all the values were also cast to a double value using the cast to double block.

Serial Communications - Transmit

The DCM_out() subsystem is designed to communicate with the DCM by sending information based on the value of the COM input. Its purpose is to allow the DCM to either monitor the real-time electrical signals of the heart (atrium and ventricle) or inspect the system's current parameter settings. This makes the subsystem critical for ensuring the DCM has the required data to make real-time decisions or configuration assessments. The overall design can be seen below in **Figure 9:**

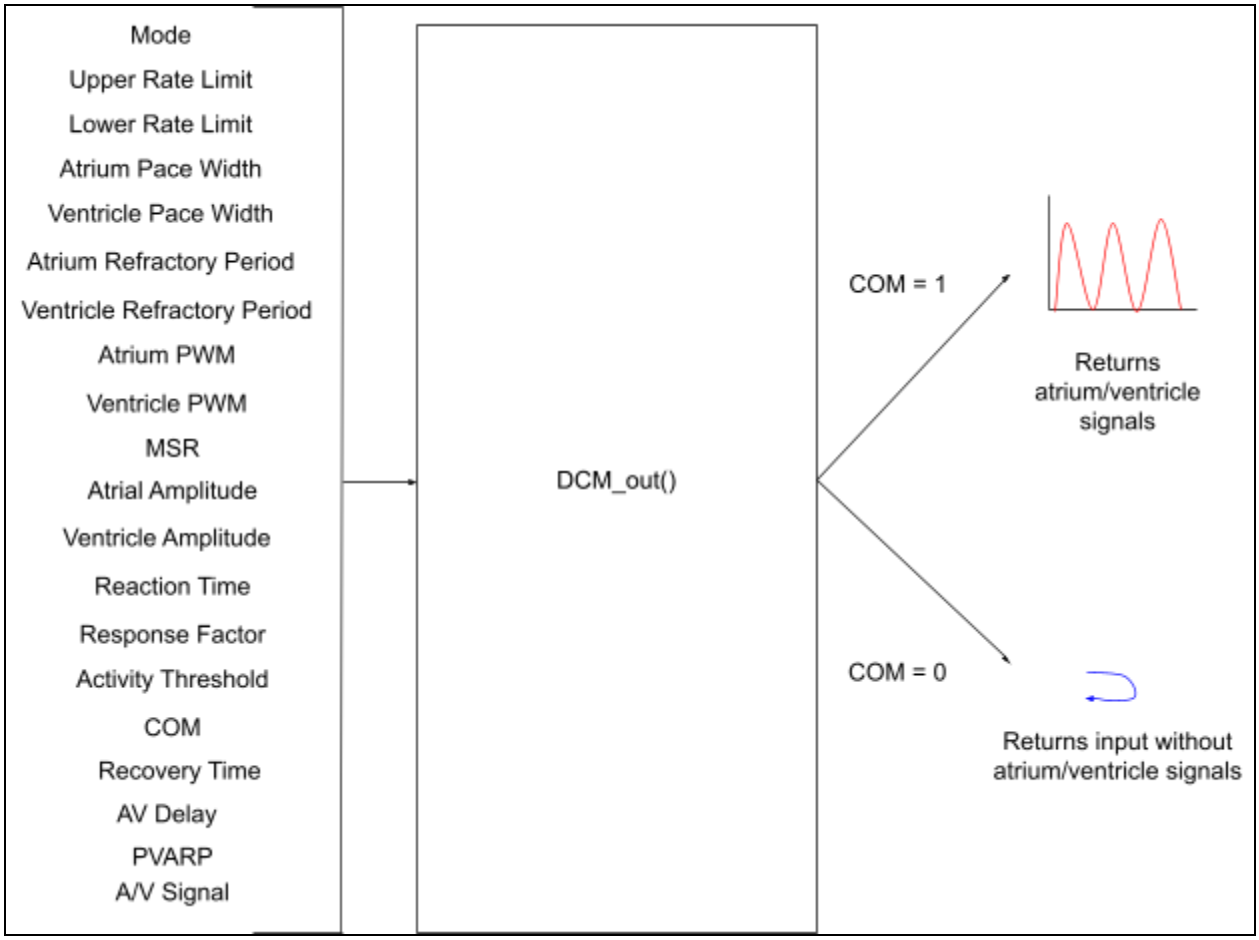

Figure 9: Overall design of DCM_out() subsystem in Simulink.

When COM = 1, the subsystem outputs the real-time atrial and ventricular signals, giving the DCM direct insight into the heart's activity. On the other hand, when COM = 0, the subsystem outputs only the current parameter settings, omitting the heart's real-time signals. These parameters include pacing widths, refractory periods, amplitudes, and other configuration details necessary for cardiac monitoring and control.

The design of the DCM_out() subsystem incorporates several Simulink components to achieve this functionality. First, the inputs to the subsystem are grouped using multiplexers (otherwise known as “mux”). A mux is a block in Simulink that combines multiple input signals into a single output vector. In this system, two separate muxes are used: one for aggregating the input parameters and another for combining the real-time atrial and ventricular signals (**Figure 10**). This approach ensures all inputs are consolidated into two distinct vectors, simplifying their downstream processing.

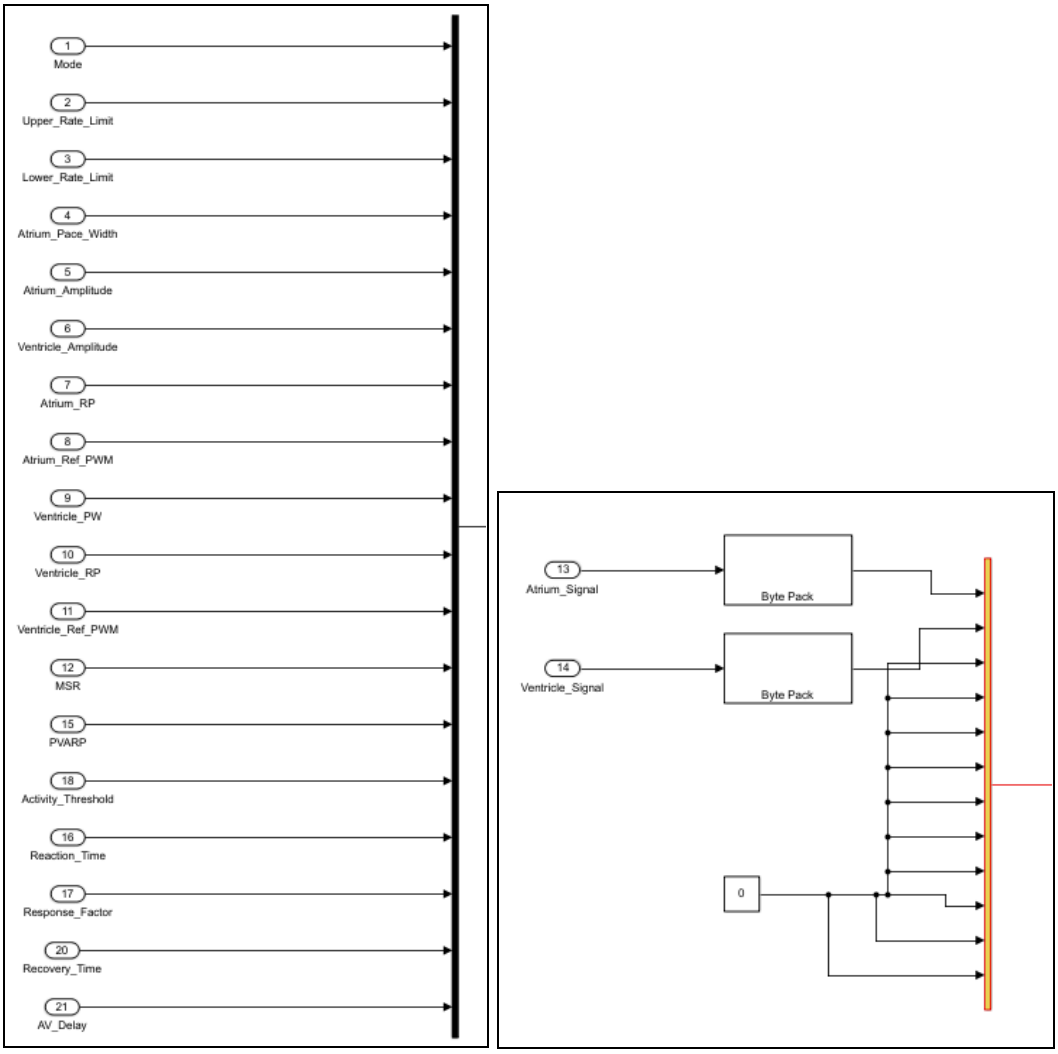

Figure 10: Multiplexers. Mux #1 is on the left, and Mux #2 is on the right.

To ensure both muxes have the same length, constant zero values are added to the smaller vector. This step is necessary because the switch in the subsystem requires both inputs to have the same dimensions to function properly. Specifically, ten zeroes are appended to the mux containing the input parameters, matching the size of the mux with the atrial and ventricular signals.

For the atrium and ventricle signals, they come from pins A0 and A1, respectively. This outputs the analog wave of the atrium/ventricle. Before the data is processed further, the atrium and ventricle signals are converted to singles. This conversion standardizes the data type, ensuring compatibility across the subsystem and improving computational efficiency. After unit conversion, the data is passed through a byte-packing block. Byte packing is a process that compresses the data into a compact format, making it suitable for efficient transmission to the DCM. It essentially converts input arrays into a smaller, serialized format that reduces the amount of memory required for communication.

The final step in the subsystem is the switch, which is controlled by the COM signal. The switch determines whether the output consists of the parameters or the atrial and ventricular signals. If COM = 0, the condition for this branch of the switch is true, and the system outputs the muxed parameters back to the DCM. On the other hand, when COM = 1, this condition is false, and the switch outputs the real-time atrial and ventricular signals instead. This behaviour ensures the DCM receives the correct type of information based on the value of COM.

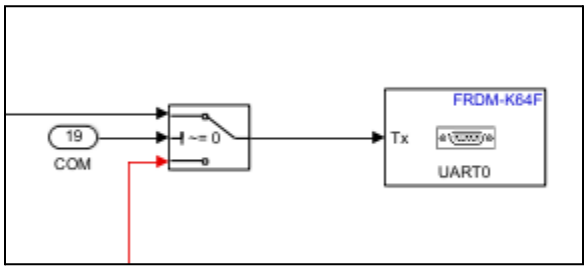

Figure 11: Switch logic. The top terminal is opened when COM = 0, and the bottom terminal is opened when COM = 1.

Mode Decision

After the input subsystem, the model moves into the Modes state flow diagram. The simulation initially enters the Mode block, which sets all the variables used by the modes to 0, as seen in **Figure 12**. This is to ensure the simulation is properly reset and capacitors C21 and C22 are discharged after changing modes. Then, it decides which pacing mode Stateflow diagram to enter based on the value of the mode variable, where 1 is AOO, 2 is VOO, 3 is AAI, 4 is VVI, 5 is AOOR, 6 is VOOR, 7 is AAIR, and 8 is VVIR. It stays in the selected mode diagram until it is changed, at which point it leaves the current mode and returns to the Mode block, where it resets, and then enters the next mode diagram based on the value of the updated mode variable.

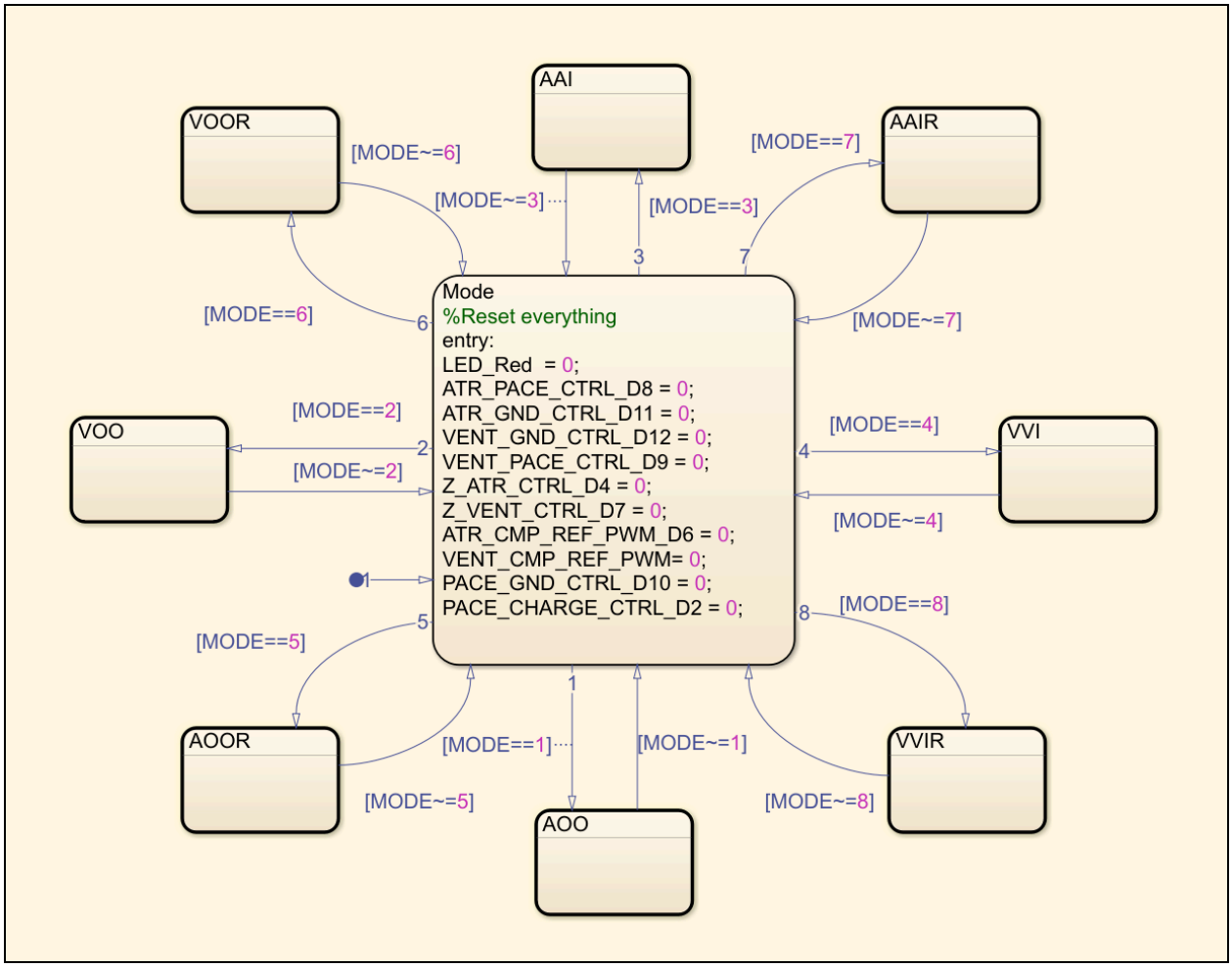

Figure 12: Logic for choosing which mode to activate (based on the MODE parameter from the input subsystem)

AOO

The AOO mode paces the atrium chambers but does not sense or respond to sensing. This means that even with a natural heartbeat, the pacemaker will asynchronously pace the atrium. The AOO state flow diagram is depicted in **Figure 13**. The first block the simulation enters is the ChargeDischarge block. Two main events happen, capacitor C21 is discharged to reset everything, and capacitor C22 is charged to prepare for the next pulse. Following a standby period, which is calculated using the lower rate limit, the simulation enters the pace block. A blue LED will turn on to indicate pacing. After the pre-defined pulse width, the simulation re-enters the ChargeDischarge block, and this entire process will loop.

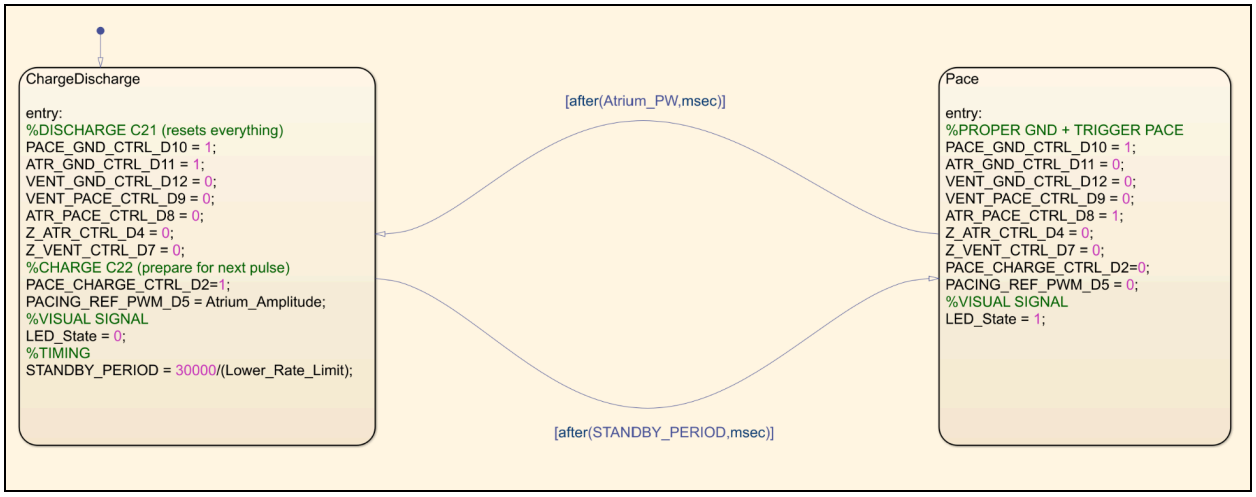

Figure 13: Stateflow diagram of AOO mode on Simulink

VOO

Similar to AOO, VOO paces the ventricle chambers asynchronously (with no sensing or responses to sensing). This mode is designed very similarly to AOO, with the main difference being the input parameter magnitudes and the active pins. This means that the simulation first enters the ChargeDischarge block, capacitor C21 discharges, and capacitor C22 charges. Following a waiting period, determined by the lower rate limit, the simulation progresses to the pacing block. At this point, a blue LED turns on to indicate that pacing is occurring. After the pulse is delivered, the simulation returns to the ChargeDischarge block to restart the cycle, repeating this process continuously.

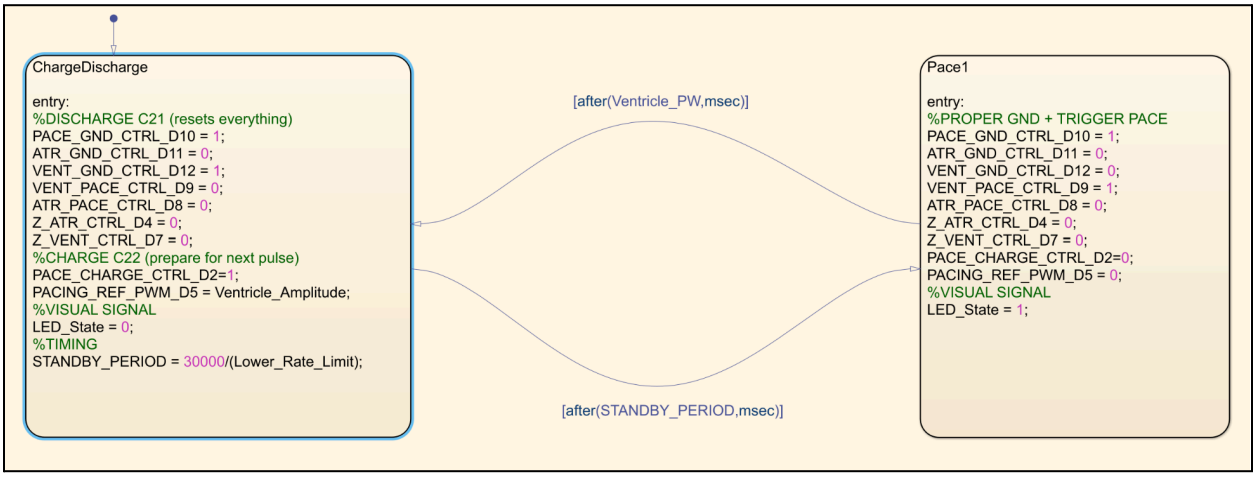

Figure 14: Stateflow diagram of VOO mode on Simulink

AAI

The AAI mode paces the atrium chambers, while also sensing these chambers and inhibiting pacing when sensing the natural heartbeat. This mode starts in the refractory block, where timing is set before moving to the sense block, where a red LED turns on. If a natural heartbeat is detected, indicated by ATR_CMP_DETECT_D0 being set to 1, the simulation returns to the refractory block. If the pin is 0, indicating no natural atrial activity, the simulation progresses to the pacing block. In the pacing block, the green LED turns off, a blue LED turns on to signal pacing, and then the cycle returns to the refractory block, with the blue LED turning off. This process continues cyclically, pacing only when no intrinsic heartbeat is detected. However, it will pace for a heartbeat below 60 ppm.

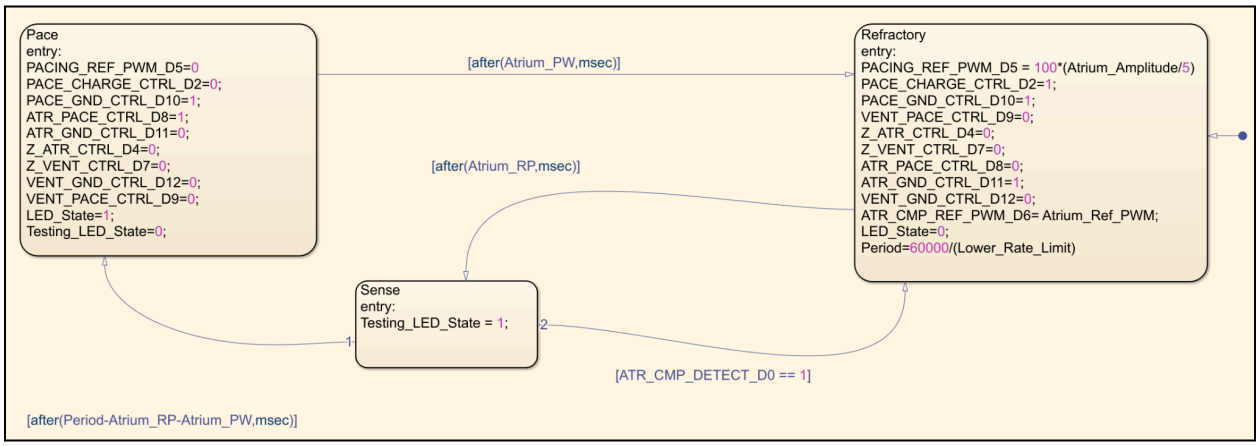

Figure 15: Stateflow diagram of AAI mode on Simulink

VVI

The VVI mode functions similarly to AAI. The difference is that if a natural ventricular heartbeat is detected, indicated by VENT_CMP_DETECT_D1 being set to 1, the simulation returns to the refractory block. If VENT_CMP_DETECT_D1 is 0, indicating no natural ventricular activity, the simulation advances to the pacing block. The green LED will turn on to indicate sensing, and the blue LED will turn on to indicate pacing.

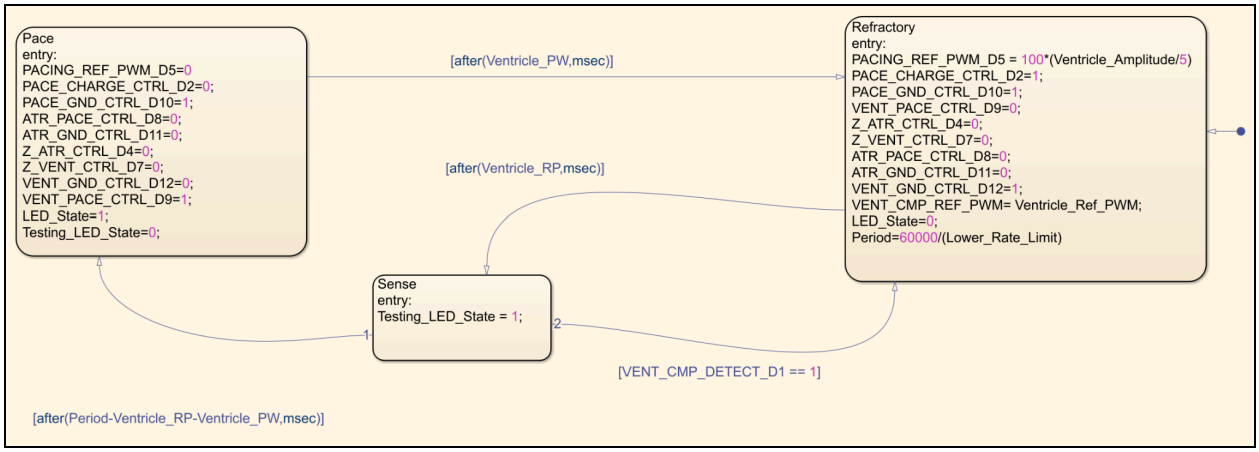

Figure 16: Stateflow diagram of VVI mode on Simulink

AOOR

The AOOR mode is a modified version of AOO. Rather than using the lower rate limit to calculate the period between pulses, it uses Desired_PR, which is a variable calculated by the Rate Regulation subsystem. When the pacemaker is detecting increasing activity, the pacing rate will gradually increase until it reaches the MSR. If it detects decreasing activity, it will decrease the pacing rate until it returns to the lower rate limit. This mode paces the atrium, and does not sense atrial activity or inhibit pacing in response to natural atrium pulses.

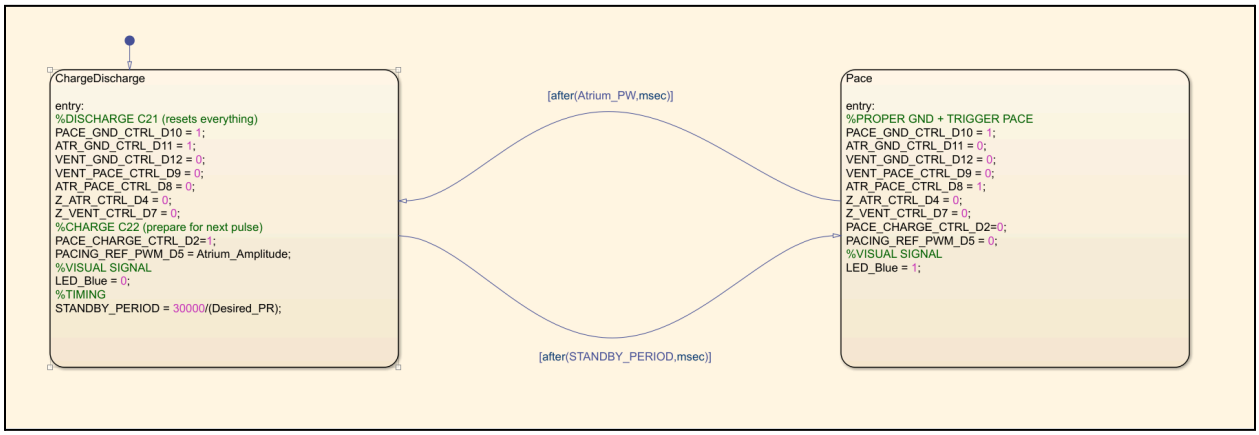

Figure 17: Stateflow diagram of AOOR mode on Simulink

VOOR

VOOR is almost identical to AOOR, except it paces the ventricle instead of the atrium. In the stateflow, the values of VENT_GND_CNTR_D12 and VENT_PACE_D9 are used instead of the atrial variables.

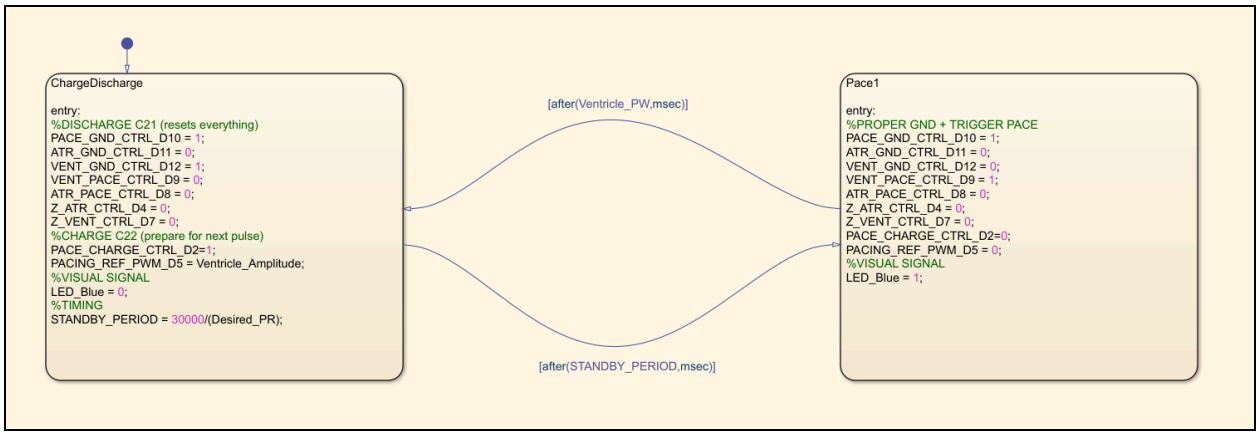

Figure 18: Stateflow diagram of VOOR mode on Simulink

AAIR

The AAIR mode is a modified version of AAI with rate adaptive pacing functionality added. The mode begins in the Refractory state, where the period is calculated using the Desired_PR, which is determined based on the accelerometer data. It then moves to the sensing block where if a natural heartbeat is detected, it returns to the Refractory block where the period is updated again. This is to ensure that the pacemaker will begin pacing if the desired pacing rate increases above the patient's natural heart rate, due to physical activity detected by the accelerometer. If a natural heartbeat is not detected within the time specified in the wait condition, the pacemaker will deliver a pace to the atrium.

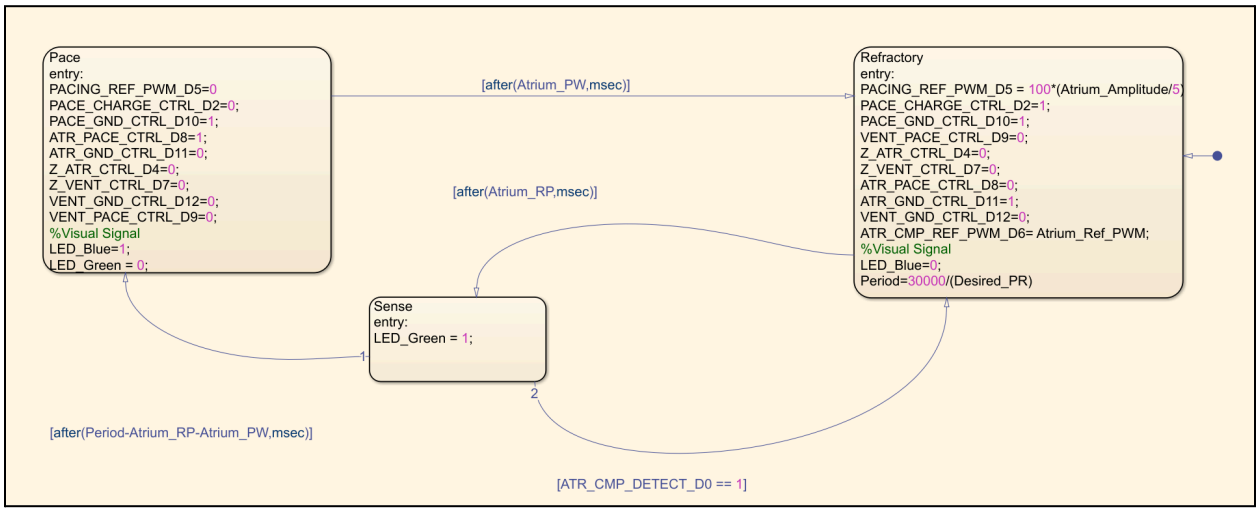

Figure 19: Stateflow diagram of AAIR mode on Simulink

VVIR

Like VOOR, VVIR is very similar to AAIR. The only difference between AAIR and VVIR is that the atrial variables are replaced with ventricular variables.

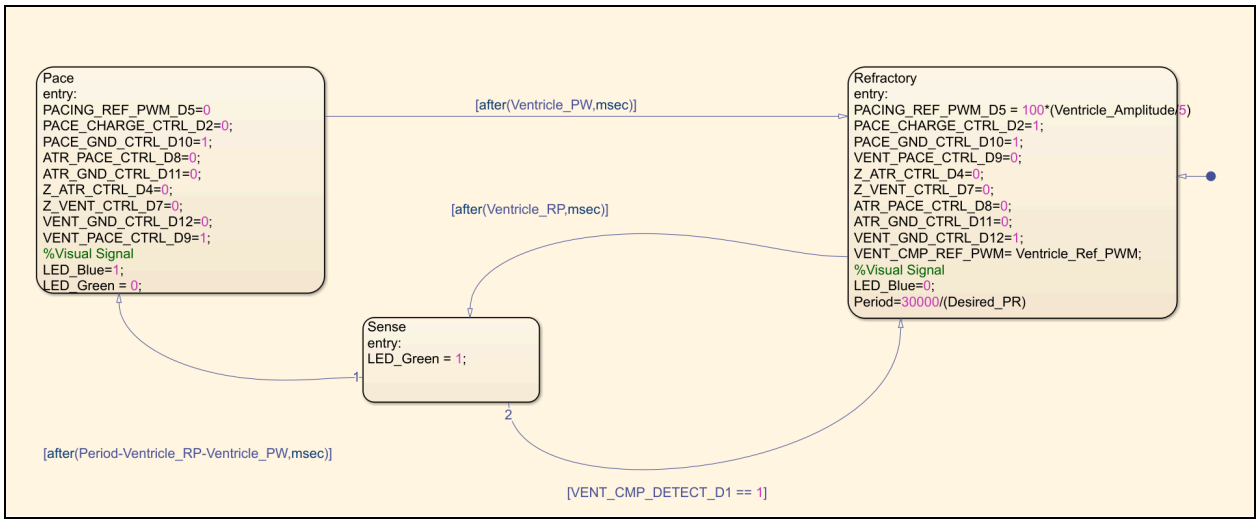

Figure 20: Stateflow diagram of VVIR mode on Simulink

Rate Adaptive Subsystem

In order to get the accelerometer data from the pacemaker and use in it the Simulink model, a 6-axes sensor block is used, as shown in **Figure 21**. This signal is then passed into Demux to decompose the vector into 3 components. The last (z) component is subtracted by 1, which was determined experimentally and was hypothesized to account for the gravitational component of the z-axis. The magnitude of the accelerometer signal is then determined by adding the square of each vector component and taking the square root, all through mathematical Simulink blocks.

The resulting magnitude is still from a raw signal, therefore it must be passed through a Moving Average block to smooth out short term fluctuations, while capturing long term trends. The sliding window in the Moving Average block takes 300 samples to calculate the average, which is then amplified by a 25 magnitude gain block. Both of these values were determined experimentally. This value is then passed as the Activity_Level variable into the MATLAB rate_adaptive block that calculates the desired pace rate.

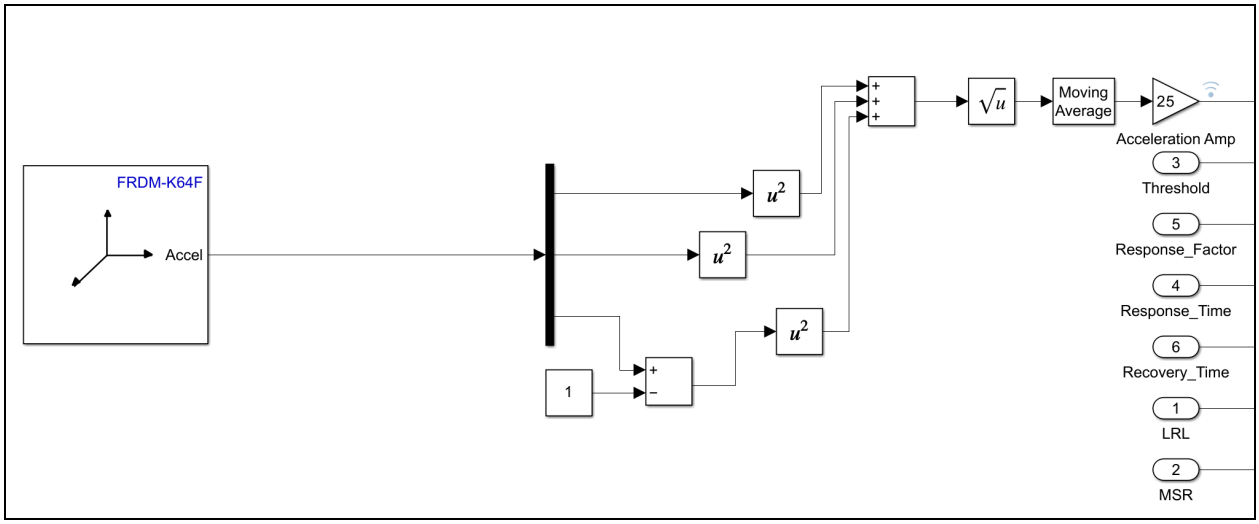

Figure 21: Accelerometer data processing in the Rate Adaptive Subsystem

A MATLAB function block in **Figure 22** is then used to calculate the desired pace rate (Desired_PR) by taking in other programmable variables shown in **Table 5**. The first if loop in **Figure 23** that tests whether Activity_Level is less than -10 was created after testing Rate Adaptive modes. The accelerometer was not responding when shaking the pacemaker. Therefore, this loop was implemented to avoid over penalizing low activity levels to ensure a visual change in the pace rate can be observed in a shorter time period. Then, the derivative of activity levels determines whether the pacemaker is increasing or decreasing. This would then determine whether the reaction time or recovery time variable should be used. The acceleration is then calculated using the appropriate time variable, the difference between the activity level and the threshold, and the response factor to determine the incrementation of the desired pace rate. A preliminary desired pace rate is then calculated by adding the acceleration to the previous pace rate. However as a safety measure, the last if loop tests whether the desired pace rate is between the allowable upper and lower pacing bounds. This ensures that the pacemaker is not sending any harmful signals to the user.

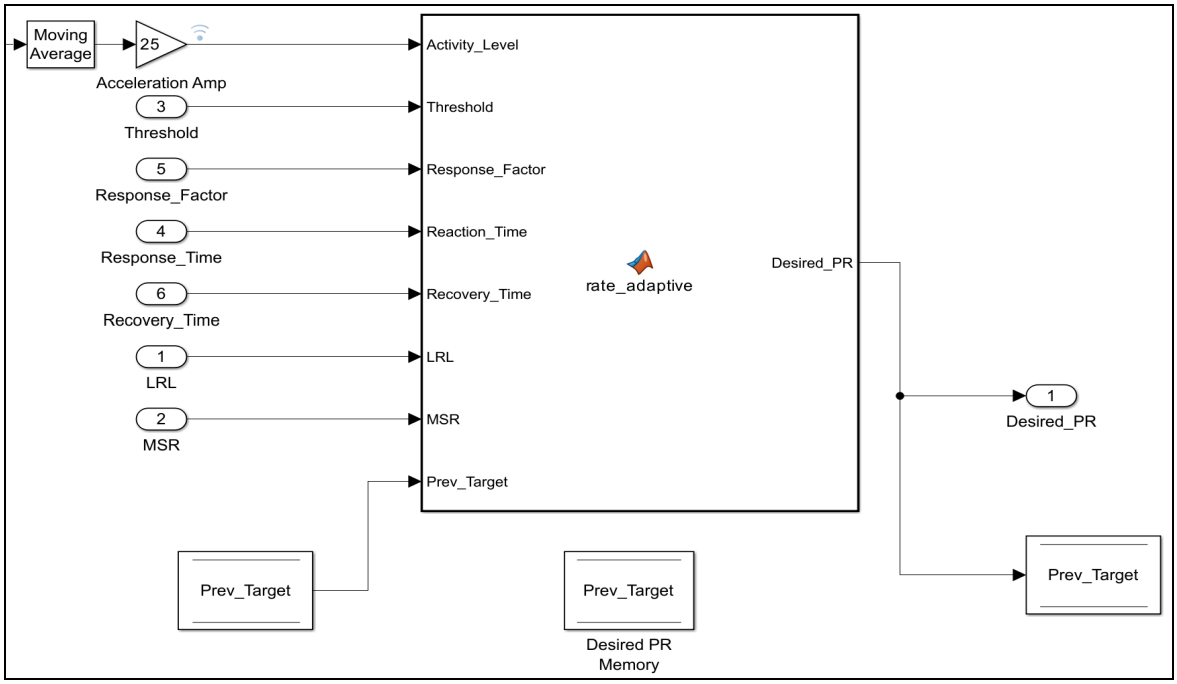

Figure 22: Rate Adaptive MATLAB function block inputs

```
% Rate adaptive starts when the activity level if greater than the threshold

Gain = 10;

Activity_Derivative = diff(Activity_Level);

Activity_Difference = Activity_Level - Threshold;

% To prevent over penalizing very low activity levels
if (Activity_Difference < -10)
    Activity_Difference = -10;
end

if (Activity_Derivative>=0) % Increasing PR
    Accel = Gain*Activity_Difference*Response_Factor/Reaction_Time;

else % Decreasing PR
    Accel = Gain*Activity_Difference*Response_Factor/Recovery_Time;

end

% Desired pace rate increases with acceleration and previous rate
Desired_PR = Accel*(1e-3)+Prev_Target;

% However, to ensure Desired PR is not outside of bounds based on Accel incrementation, final check:
if (Desired_PR<LRL)
    Desired_PR = LRL;

elseif (Desired_PR>MSR)
    Desired_PR = MSR;

end
```

Figure 23: Rate Adaptive MATLAB function block code logic

Output Subsystem

The output subsystem is designed to manage and control the various hardware components of the pacemaker, as shown in the figure. Each pin on the FRDM-K64F board corresponds to a specific control signal used in the pacemaker's operation. These signals manage functions such as charging the pacing capacitor, detecting atrial and ventricular activities, and controlling the pacing events. For example, the ATR_PACE_CTRL (D8) and VENT_PACE_CTRL (D9) control the discharge of the pacing capacitor through the atrium and ventricle, respectively, initiating the pacing process. Additionally, signals like the PACE_GND_CTRL (D10) ensure proper current flow during the pacing event. LED indicators are used to provide visual feedback for pacing and safety signals, such as the blue LED for pacing events. The output subsystem thus translates the high-level pacemaker logic into physical actions that affect the hardware, ensuring that pacing, sensing, and safety checks are executed appropriately.

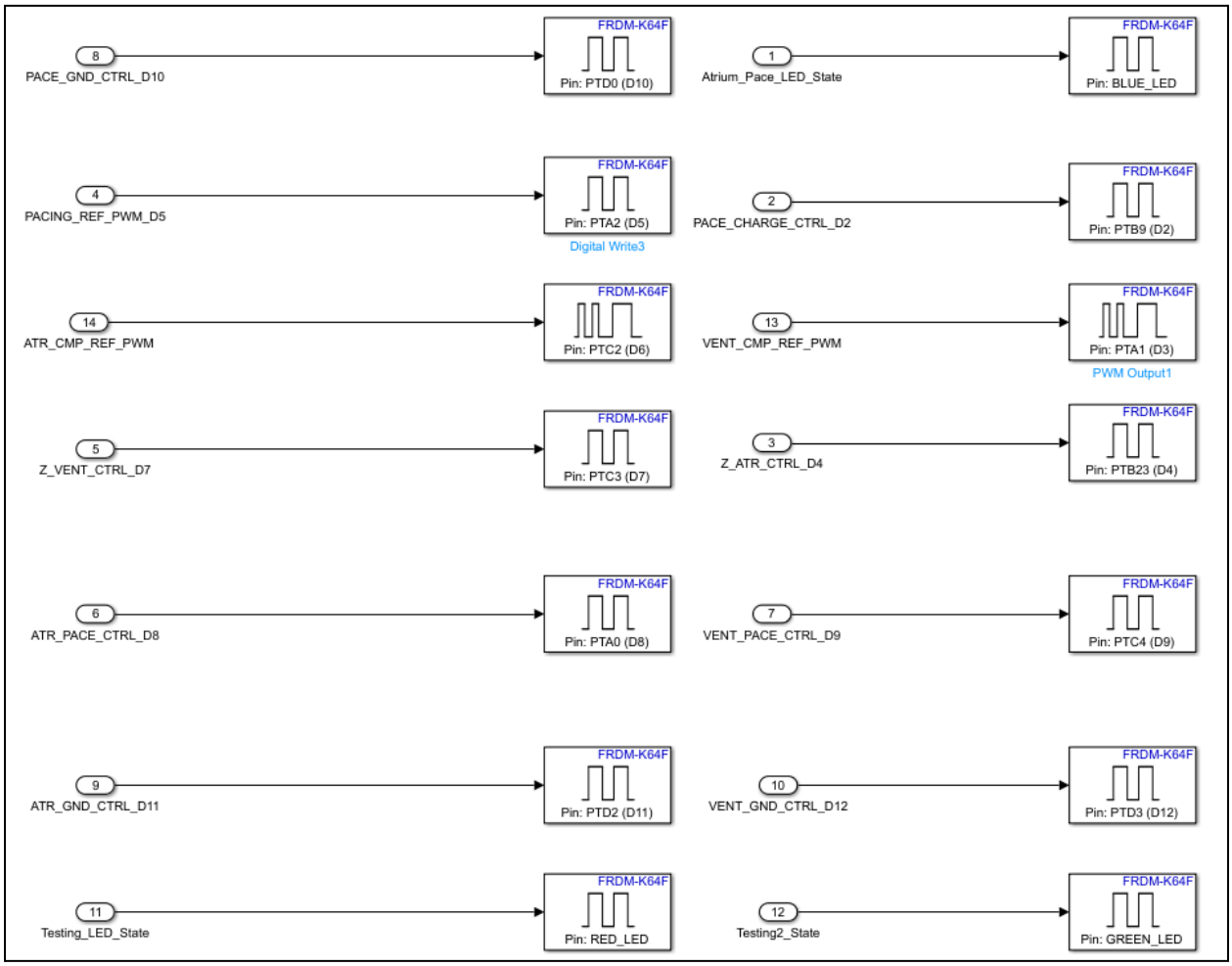

Figure 24: Output parameter subsystem that allows for sensing and pacing.

DCM Design:

Design Considerations

This section outlines the design of the Device Controller-Monitor (DCM), detailing how its user interface and functionalities work together to create a straightforward and adaptable system. Built using PyQt5 in Python, the DCM handles user authentication, pacing mode selection, and parameter adjustment, while keeping each functionality separate and manageable. This setup makes the interface intuitive and easy to navigate.

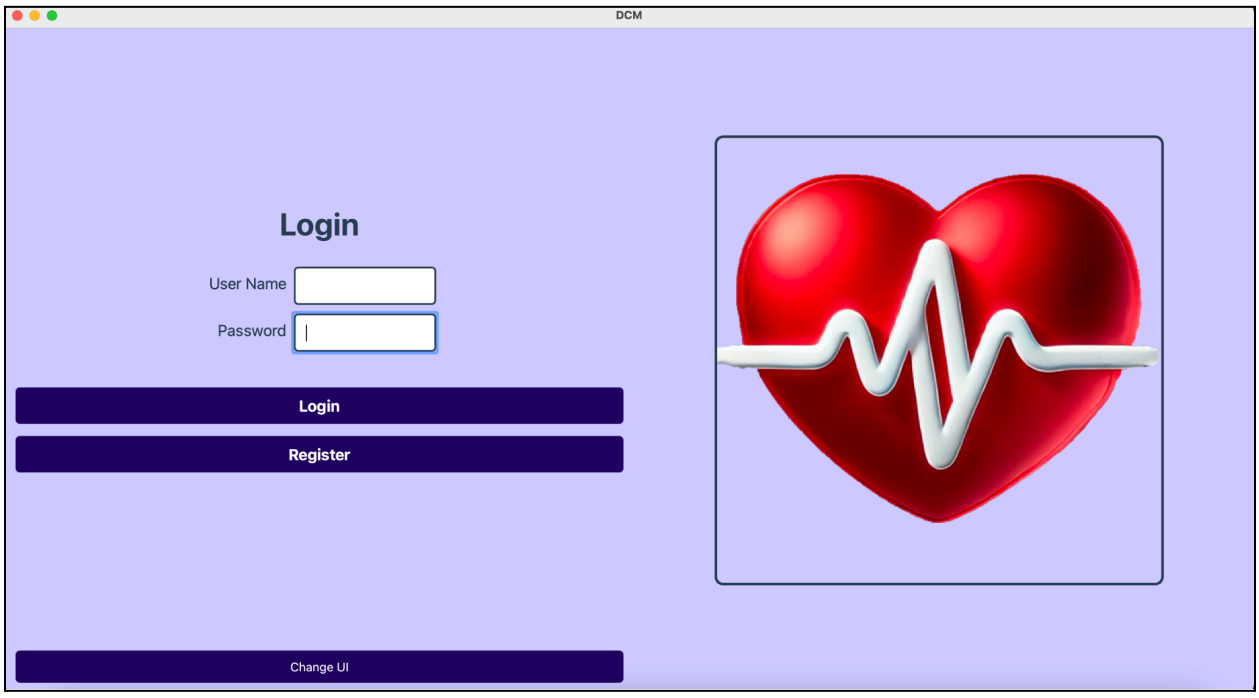

Figure 25: Home Login Page

The DCM begins with login and registration screens that control access to the main pacing mode functionalities. Registered users can log in and reach the main page, where a dropdown allows them to select from the four pacing modes available: AOO, VOO, AAI, and VVI. Each mode displays its parameters—such as lower and upper rate limits, amplitude, and pulse width—using sliders that are bound to the given ranges. The stacked widget structure loads only the selected mode's parameters at a time, keeping the layout clean and focused. This structure makes it easy to add new modes or adjust parameters for specific modes without affecting the interface of other pacing modes. Settings are saved locally, meaning users find their configurations intact even after logging out.

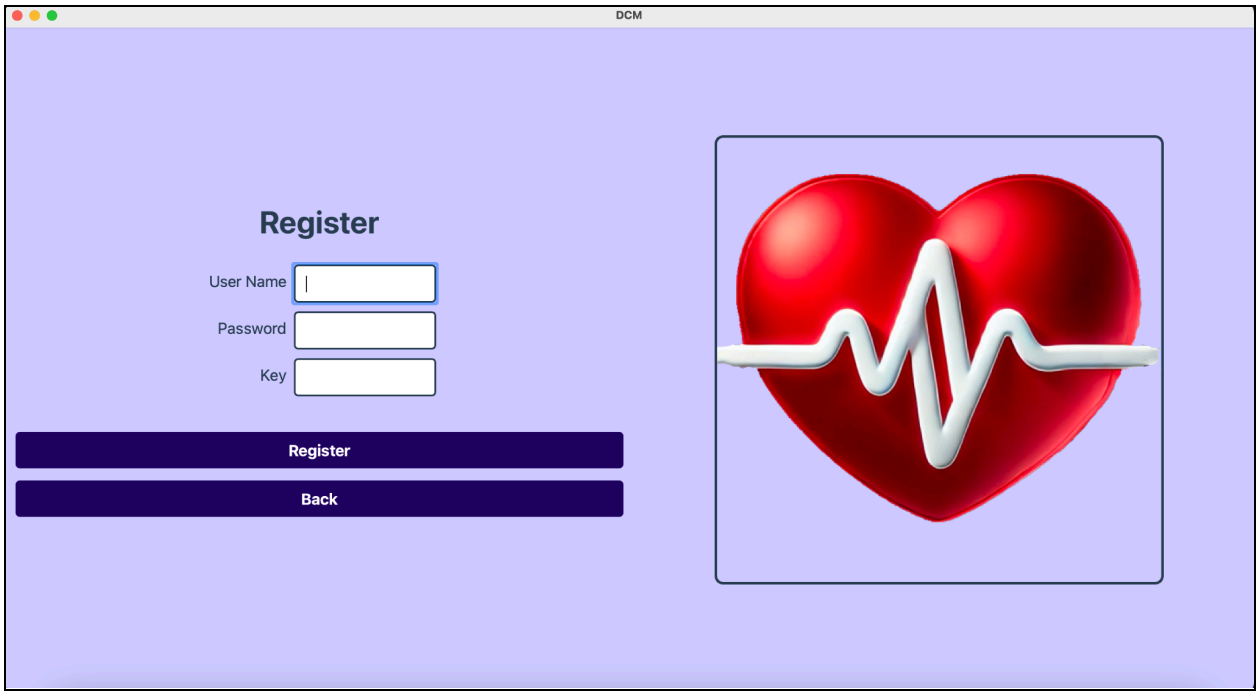

Figure 26: Registration Page with the Key Security

Navigation between screens is simple, keeping users on track as they move from login to registration and then to the main control page. Registration requires a doctor-issued key for added security, ensuring that only authorized patients can create accounts. Real-time feedback is a priority: status indicators show device connectivity and readiness, allowing users to verify the system's status at a glance—essential for any medical application requiring consistent monitoring.

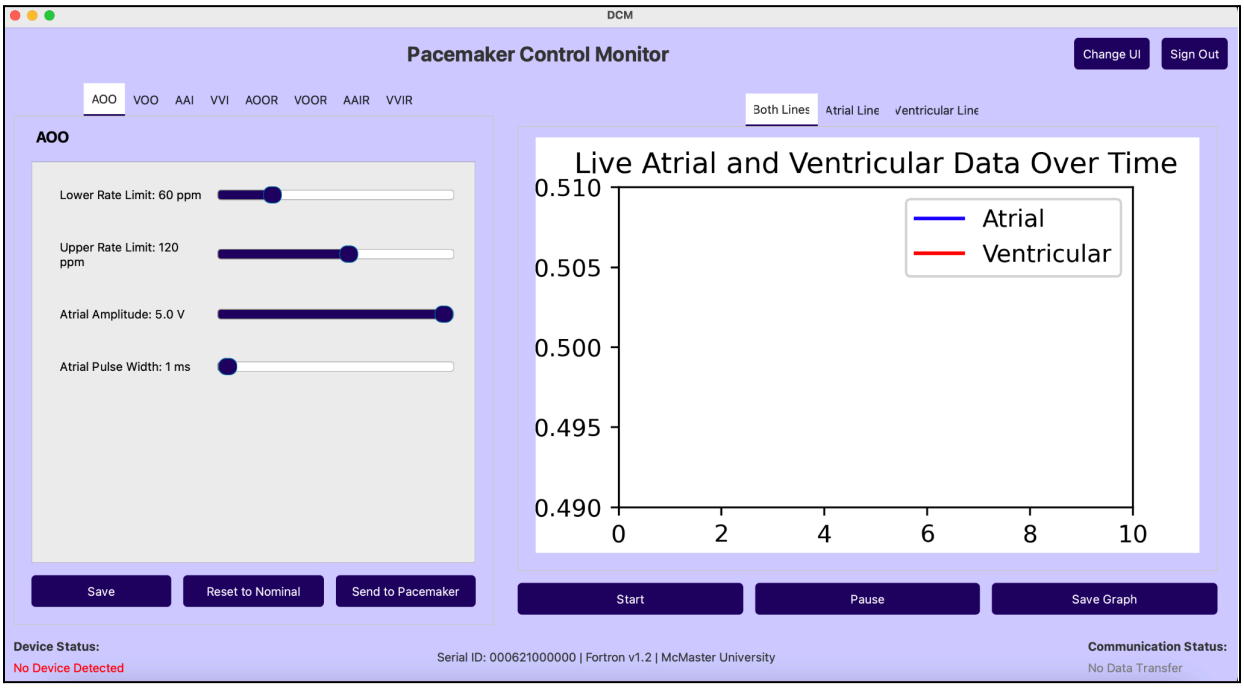

Figure 27: Main Page Set to Mode Select AOO

As you see with AOO and VOO as well as the rest of the modes, all parameters are available. Graphing data and button functionality are not shown as they are in the test cases

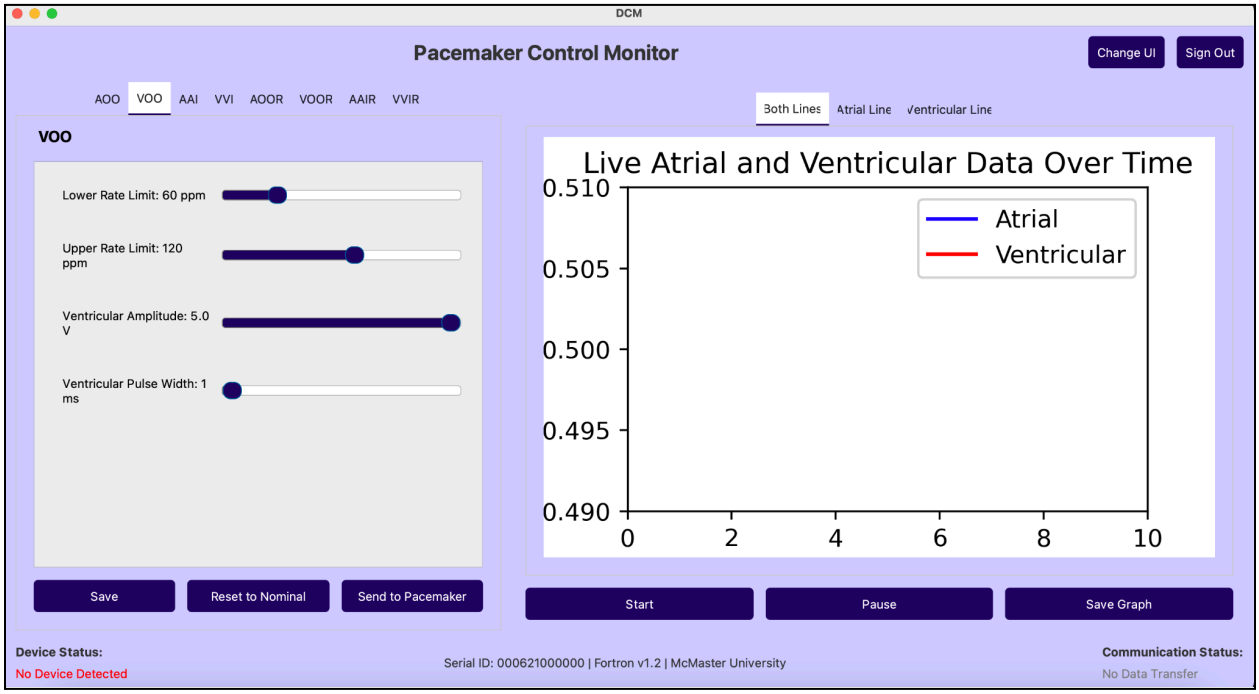

Figure 28: Main Page Set to Mode Select VOO

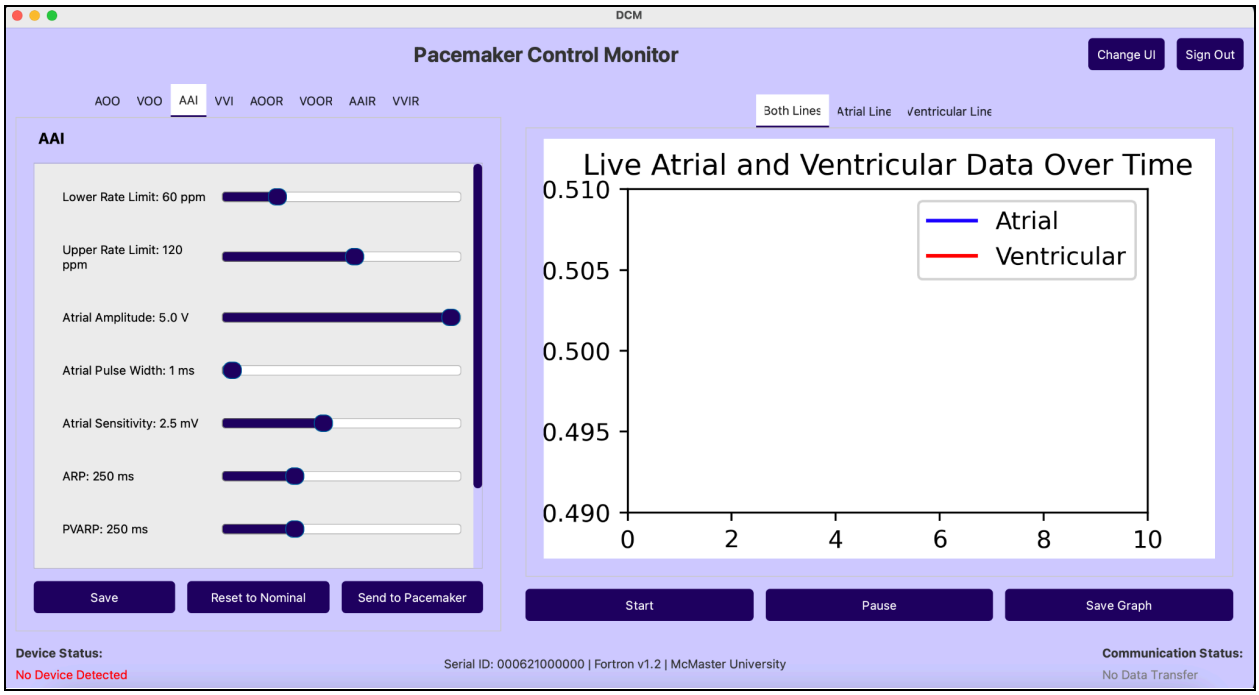

Figure 29: Main Page Set to Mode Select AAI

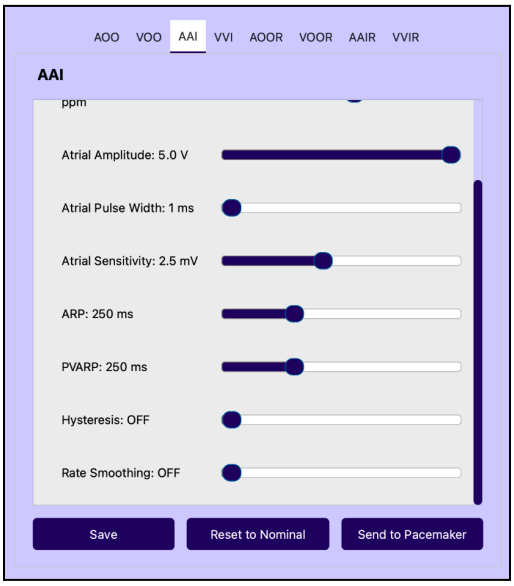

Figure 30: Main Page Set to Mode Select AAI, remaining parameters

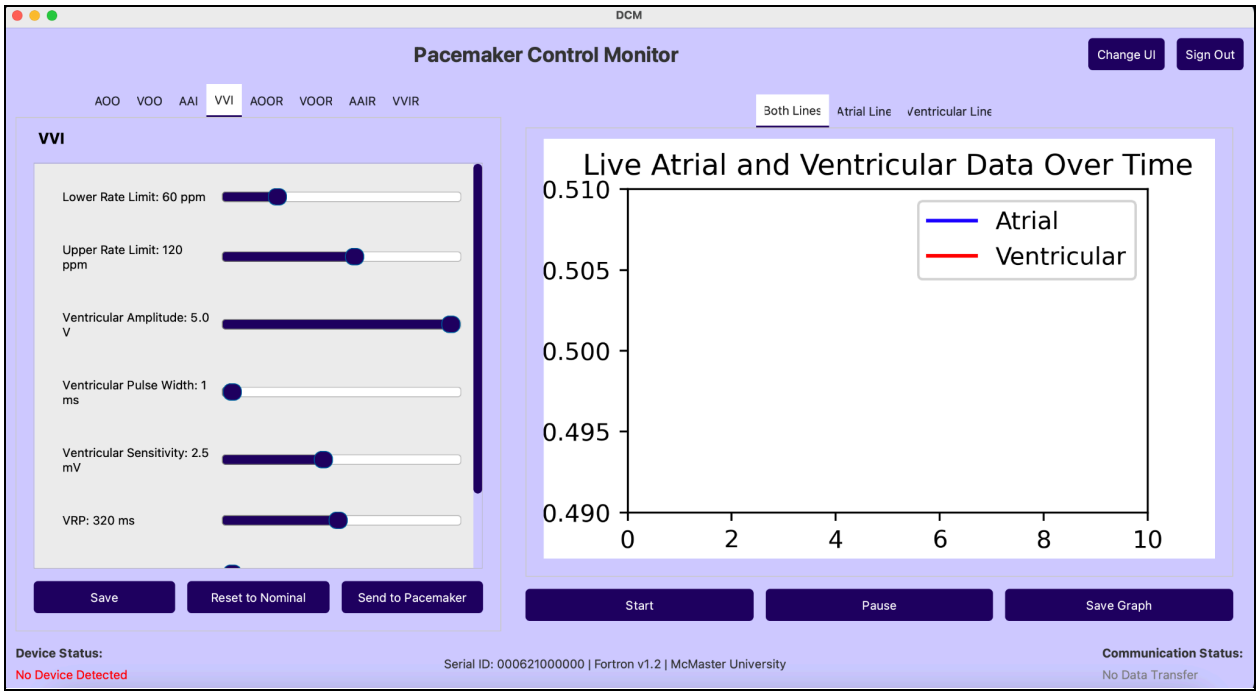

Figure 31: Main Page Set to Mode Select VVI

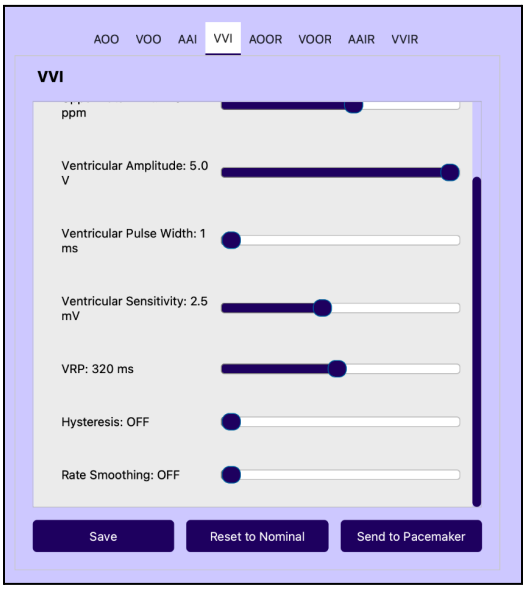

Figure 32: Main Page Set to Mode Select VVI, remaining parameters

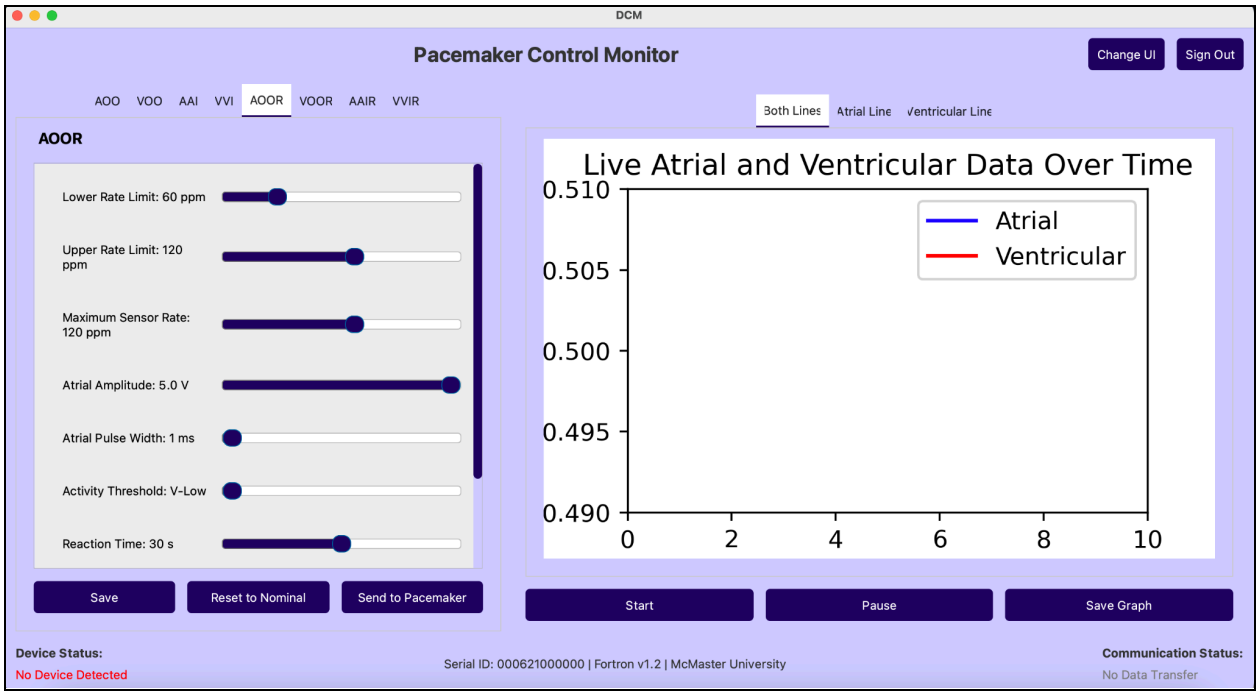

Figure 33: Main Page Set to Mode Select AOOR

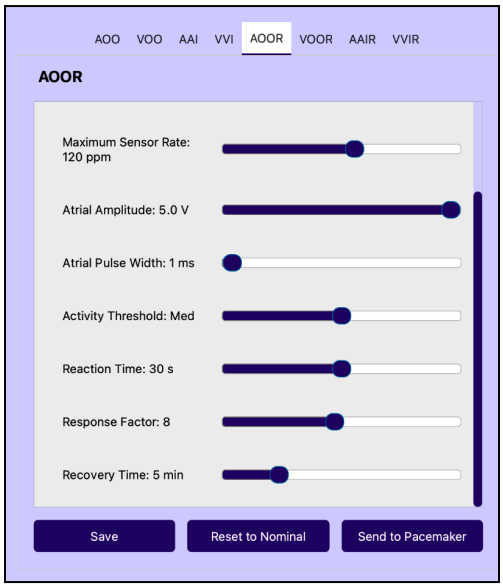

Figure 34: Main Page Set to Mode Select AOOR, remaining parameters

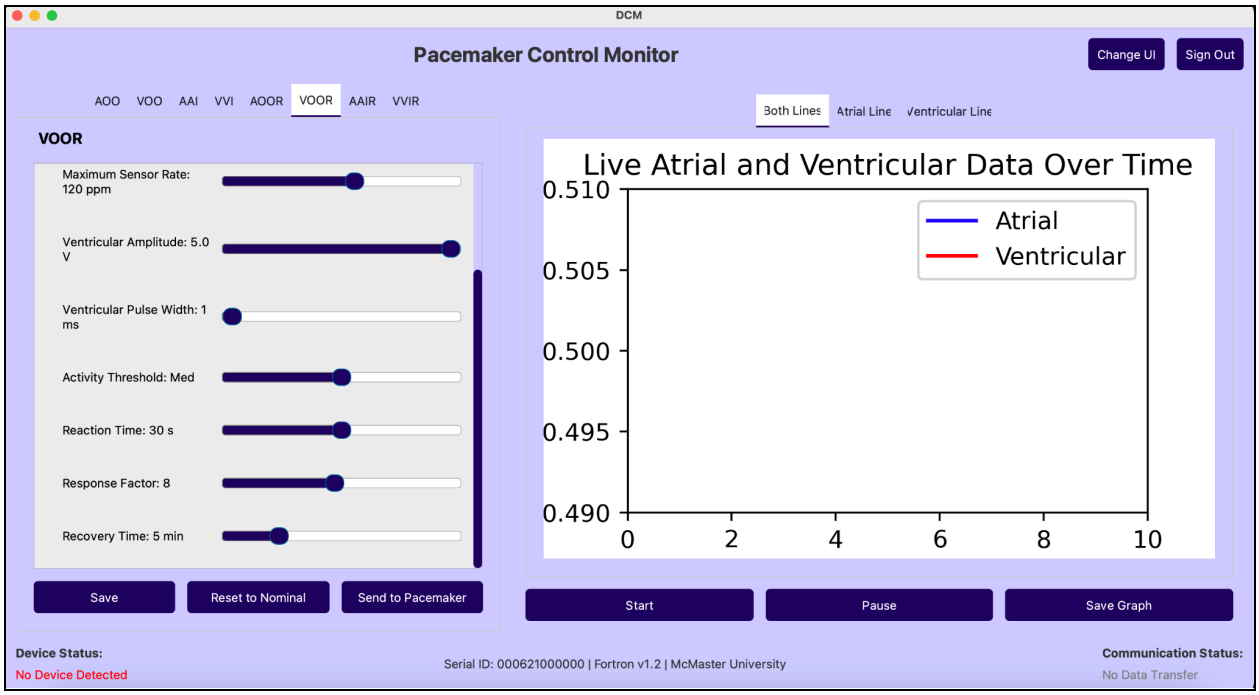

Figure 35: Main Page Set to Mode Select VOOR

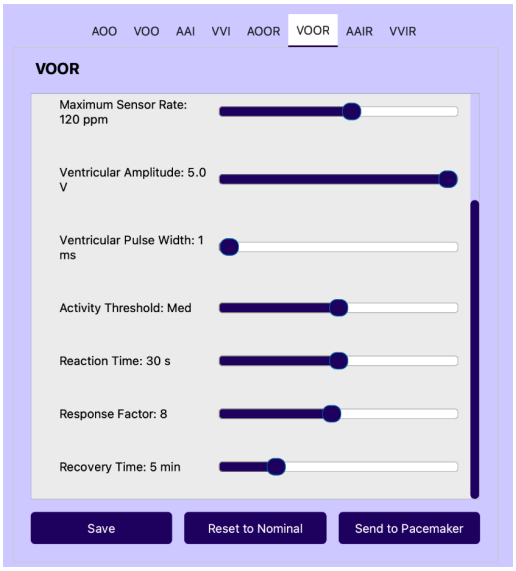

Figure 36: Main Page Set to Mode Select VOOR, remaining parameters

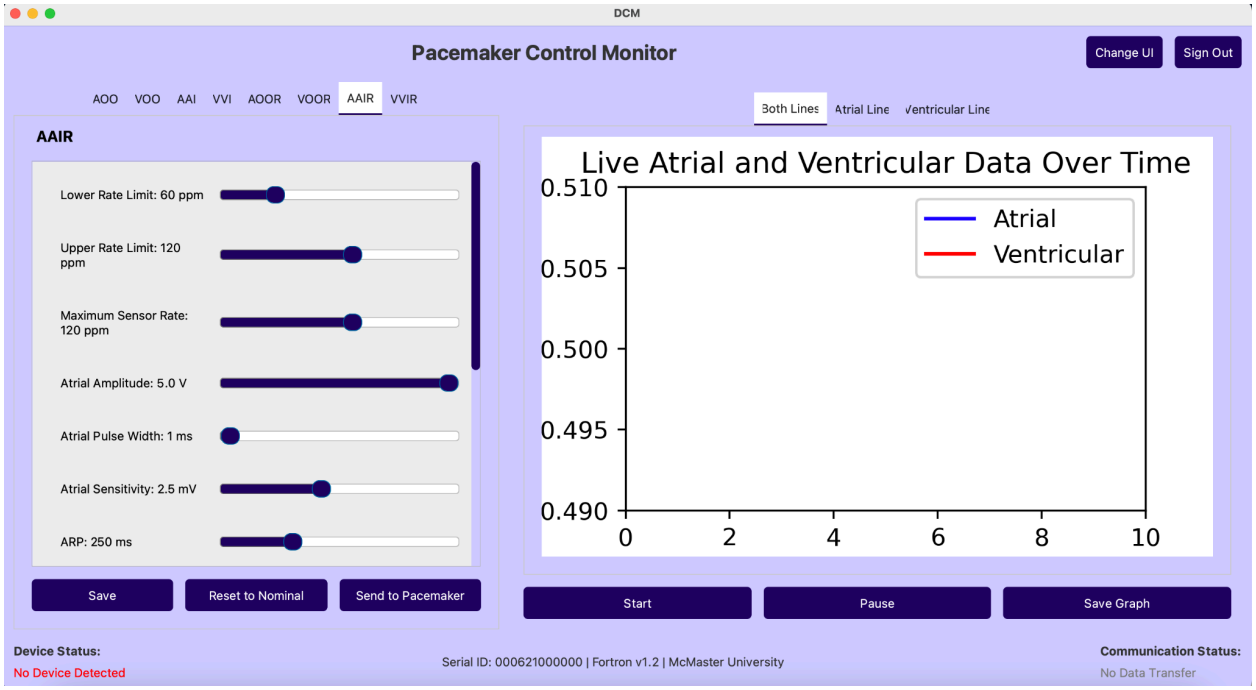

Figure 37: Main Page Set to Mode Select AAIR

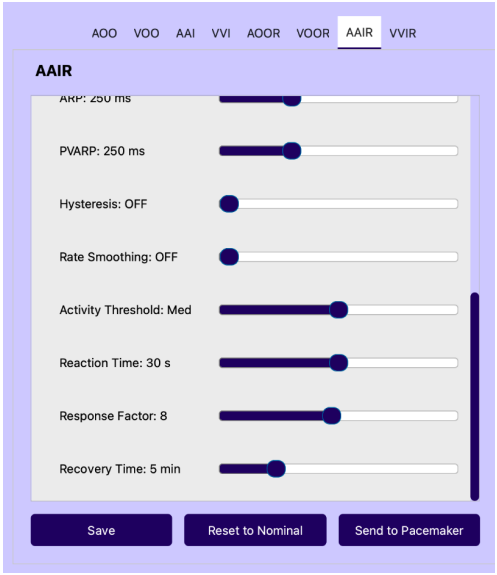

Figure 38: Main Page Set to Mode Select AAIR, remaining parameters

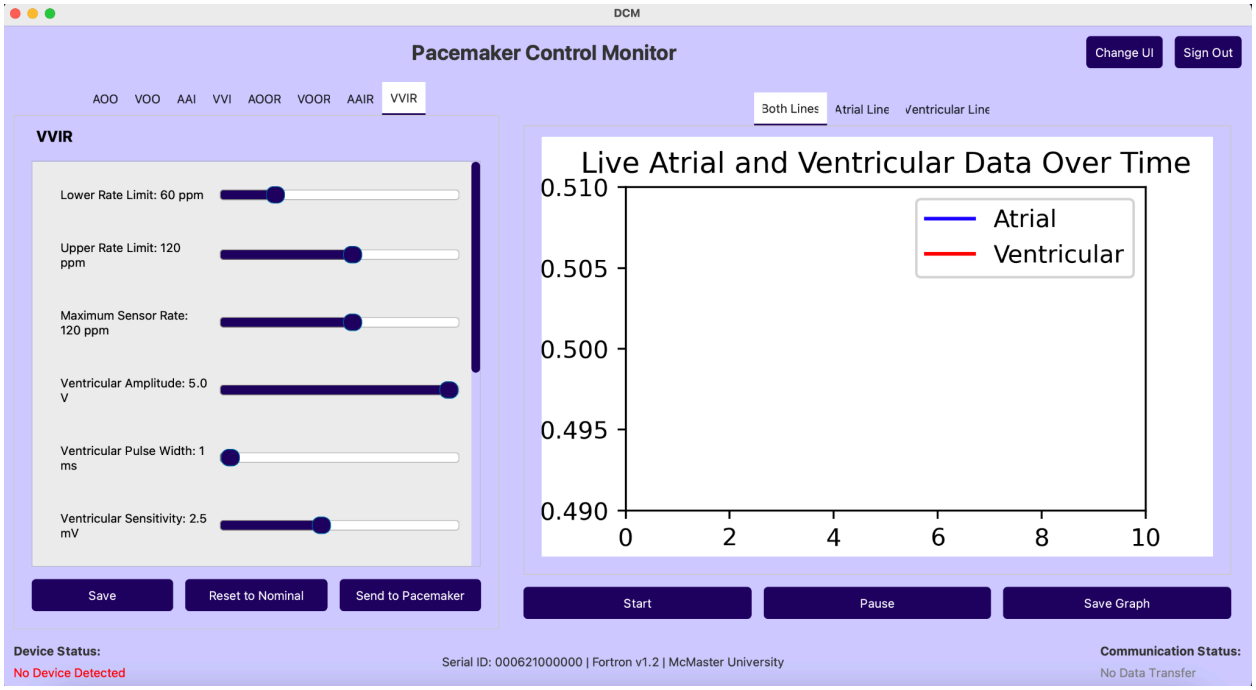

Figure 39: Main Page Set to Mode Select VVIR

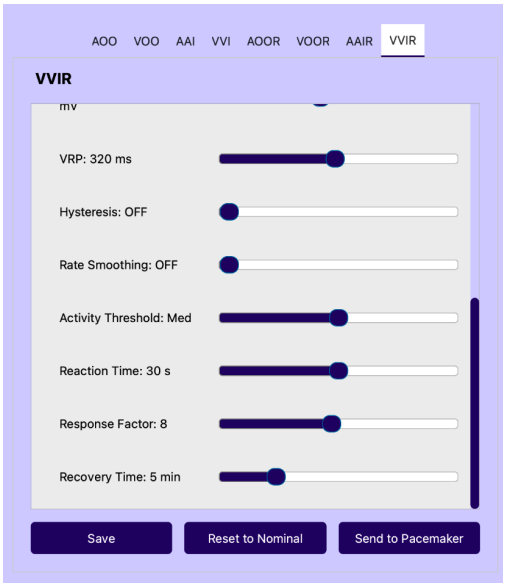

Figure 40: Main Page Set to Mode Select VVIR, remaining parameters

The design is built for adaptability and maintenance. Functions are grouped logically, with each feature (like login, mode selection, or parameter adjustment) managed independently, meaning changes in one area don't affect others. This modularity aligns with project requirements and keeps the DCM interface responsive, expandable, and ready for future updates or new features. This section explores each of these elements in detail, showing how they come together to meet the needs of a reliable pacemaker management interface.

Class Diagram

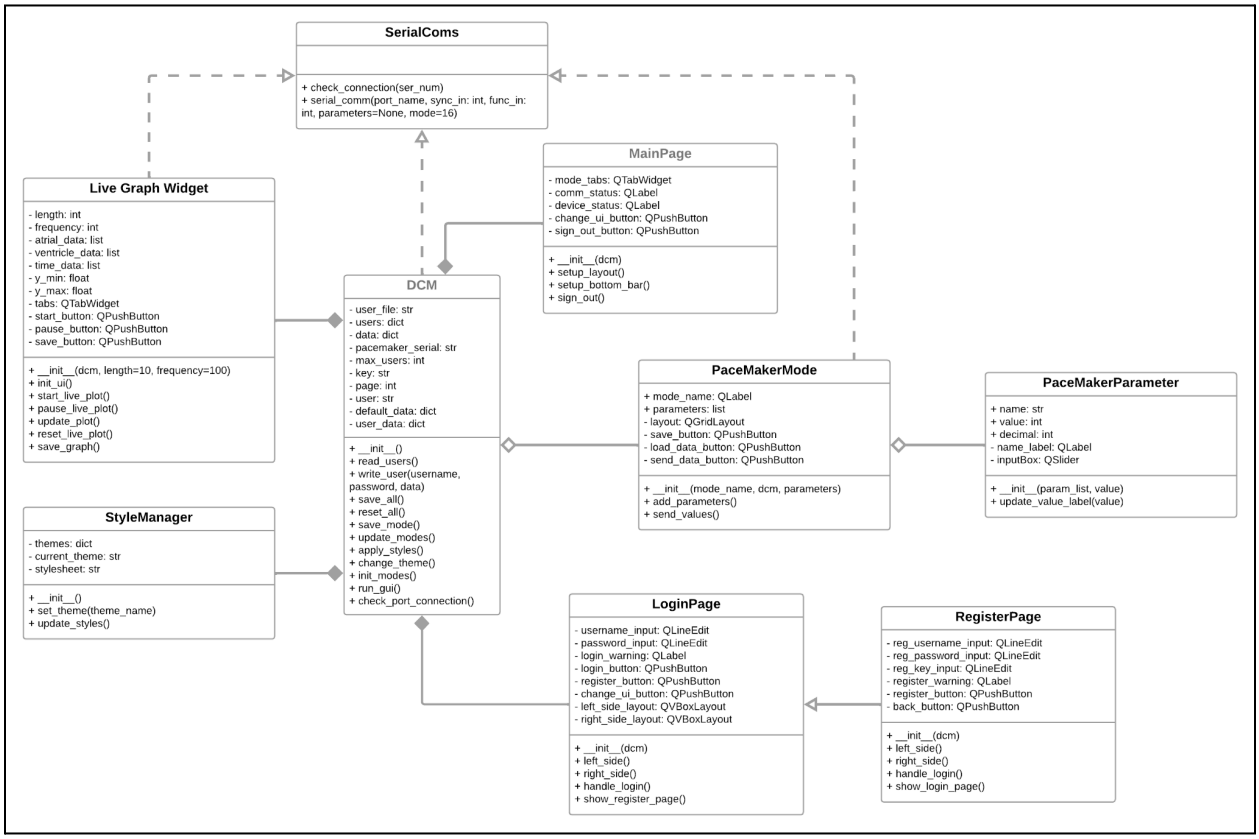

Figure 41: Class diagram for DCM

Classes

Table 7: Class black box behaviour

Class	Description
DCM	The main application class for managing the GUI flow, user authentication, and pacing mode settings. Handles reading/writing user data, orchestrates layout switching (login, registration, and main pages), and interacts with hardware using SerialComs
MainPage	the main interface page, displaying pacing mode options and their programmable parameters. Provides user interaction elements like combo boxes, status indicators, and control buttons for saving/loading configurations.
PacemakerMode	Represents a specific pacing mode with its associated programmable parameters. Organizes parameters within a grid layout and provides options to save, load, and send data to the pacemaker through SerialComs
PaceMakerParameter	Manages an individual parameter setting within a pacing mode, including the parameter's name, adjustable slider, and label displaying the current value.
LoginPage	The login interface, allowing users to enter their

	credentials and log into the DCM application. Provides options for switching to the registration page.
RegisterPage	The registration interface, allowing new users to create an account. Collects a username, password, and registration key, validates inputs, and saves the new account to the user data file if successful.
LiveGraphWidget	Displays real-time graphs for pacing and cardiac activity data. Handles live updates and user interactions like starting, pausing, and saving graphs. Depends on SerialComs for device data. Owned and managed by DCM
StyleManager	Manages the applications themes and styling. Updates the GUI appearance dynamically and supports multiple themes. Owned and controlled by DCM
SerialComs	A utility class for managing serial communication with the pacemaker device. Provides methods for checking the connection as well as sending and receiving data. Used by DCM, PacemakerMode and LiveGraphWidget

Methods

Table 8: Method black box behavior

Class	method	Description
DCM	__init__()	Initializes the DCM application, setting up essential properties and reading user data from users.txt. This method starts by setting up a QMainWindow with a title, style, and layout. Initializes the logo as a QPixmap object and loads user data by calling read_users(). Configures default user-related attributes, including page, pages, max_users, and key, and creates a dictionary of pacing modes (self.mode) and user data (self.user_data). Calls run_gui() to set up the main interface. **Args**: None. **Returns**: None.

		Attributes Modified: self.logo, self.user_file, self.users, self.data, self.page, self.pages_stacked_widget, self.max_users, self.key, self.mode, self.user, self.user_data.
	read_users()	Reads user credentials and data from the users.txt file. Opens the file in read mode, and for each line, splits the string by : to separate username, password, and saved_data. Maps usernames to passwords in self.users and maps each username's saved data (as lists of integer values) to self.data. **Args**: None. **Returns**: None. **Attributes Modified**: self.users, self.data.
	write_user(username, password, data)	Writes a new user entry to users.txt in the format username:password:data. The data parameter is expected as a list of strings, with each string representing a saved mode configuration. The function opens the file in append mode, writes the entry, and then calls read_users() to update self.users and self.data. **Args**: username (str): The new user's username. password (str): The new user's password. data (list): List of pacing mode values to save. **Returns**: None. **Attributes Modified**: self.users, self.data.
	save_all()	Saves the current parameters for all pacing mode by retrieving values from self.pacemaker_modes. Formats data as a list of strings where each entry corresponds to a mode's settings. Calls write_user() to save the data to users.txt under the current username. **Args**: None. **Returns**: None. **Attributes Modified**: None, although it calls write_user() to update user information on file.
	reset_all()	Resets all pacing modes to their default parameter values by calling reset_parameters() on each PaceMakerMode in

		self.pacemaker_modes. **Args**: None. **Returns**: None. **Attributes Modified**: None, although it updates PacemakerMode Instances directly.
	save_mode()	Saves the current pacing mode's parameters into self.user_data. Updates the users.txt file by calling write_user(). **Args**: None. **Returns**: None. **Attributes Modified**: self.user_data (stores the current pacing mode settings).
	update_modes()	Updates the pacing modes in self.pacemaker_modes by retrieving current parameter values. **Args**: None. **Returns**: None. **Attributes Modified**: Indirectly updates PaceMakerMode instances through method calls
	apply_styles()	Applies the stylesheet from StyleManager to the application by setting self.styleSheet. **Args**: None. **Returns**: None. **Attributes Modified**: self.styleSheet
	change_theme()	Cycles through available themes in StyleManager and applies the next theme. Updates self.styleSheet dynamically. **Args**: None. **Returns**: None. **Attributes Modified**: self.styleSheet
	check_port_connection()	Checks if the pacemaker device is connected via the SerialComs.check_connection() function. Updates the device status indicator in MainPage with success or failure. **Args**: None. **Returns**: None. **Attributes Modified**: self.styleSheet
	run_gui()	Loads the appropriate page (LoginPage, RegisterPage, or MainPage) into the main window based on self.page. Initializes self.pages with instances of LoginPage, RegisterPage, and MainPage. Sets up main_container with the

		selected layout for the page and displays it in the main application window. **Args**: None. **Returns**: None. **Attributes Modified**: self.pages
Mainpage	__init__(dcm)	Initializes the main interface for the pacemaker. This sets up the graphical user interface (GUI) for controlling and monitoring the pacemaker device. The GUI includes the main layout, a title bar with a "Change UI" button for theme changes and a "Sign Out" button, and mode tabs to switch between pacemaker modes. It also includes real-time data visualization and status indicators for device and communication. **Args**: dcm (DCM): An instance of the DCM class. **Returns**: None. **Attributes Modified**: self.params_stacked_widget, self.mode_tabs, self.comm_status, self.device_status.
	setup_layout()	Sets up the main interface layout by organizing the top bar, data visualization area, and bottom bar. The top bar includes a bold title, a "Change UI" button linked to the dcm.change_theme method, and a "Sign Out" button connected to the sign_out() method. The data visualization area includes a QTabWidget for displaying different pacemaker modes, dynamically populated from dcm.pacemaker_modes, and a graph widget (dcm.graph) for real-time data visualization. The graph occupies a prominent position to ensure clarity. The method concludes by delegating the construction of the bottom bar to the setup_bottom_bar() method. **Args**: None. **Returns**: None. **Attributes Modified**: None.

	setup_bottom_bar()	Constructs the bottom section of the interface, displaying device metadata and status indicators. It organizes three components: the "Device Status" section, which shows whether a pacemaker device is detected, the "Communication Status" section, which reflects the status of data transfer, and a metadata box displaying the pacemaker's serial number and software version. The device and communication statuses are initialized with default values ("No Device Detected" and "No Data Transfer," respectively) and styled with colors to indicate status (e.g., red for errors, grey for inactivity). These elements are arranged horizontally to maintain a clean and balanced appearance. **Args**: None. **Returns**: None. **Attributes Modified**: Self.comm_status, self.device_status
	sign_out()	Logs out the user by resetting the interface to its default state and returning to the login page. This method resets the graph widget (dcm.graph) by clearing live data and switching to the default tab. Similarly, the mode selection tabs (self.mode_tabs) are reset to their initial state. The dcm.page attribute is set to 0, indicating a return to the login page, and the GUI is restarted by calling dcm.run_gui(). **Args**: None. **Returns**: None. **Attributes Modified**: Self.comm_status, self.device_status
PaceMakerMode	__init__(mode_name, dcm, parameters)	Initializes the interface for a specific pacemaker mode. It sets up the mode name label and dynamically generates widgets for all associated parameters using the PaceMakerParameter class. These parameters are displayed in a vertical layout, along with buttons for saving, resetting, and transmitting settings. **Args**: mode_name (str): Name of the pacing mode. dcm (DCM): DCM instance. parameters (list): List of parameter configurations. **Returns**: None.

		Attributes Modified: self.dcm, self.name, self.parameters, self.layout
	init_layout(self)	Builds the layout for the mode, adding buttons for saving, resetting, and transmitting parameter values. This method organizes the buttons horizontally and ensures that they are displayed below the parameter widgets. **Args**: None **Returns**: None. **Attributes Modified**: self.layout
	add_parameters()	Adds each PaceMakerParameter to the layout for the current pacing mode. It iterates through self.parameters and places each parameter widget in the grid. Sets up additional elements, including a logo and action buttons. **Args**: None. **Returns**: None. **Attributes Modified**: self.layout
	update_parameters()	Updates the sliders and labels of all parameters based on the current user data stored in the dcm object. This ensures the displayed values match the most recent user settings. **Args**: None. **Returns**: None. **Attributes Modified**: None
	reset_parameters()	Resets all parameters to their default values, as specified in the dcm.default_data object. This method is typically used when the user selects "Reset to Nominal." **Args**: None. **Returns**: None. **Attributes Modified**: None
	transmit _mode	Validates and transmits the current mode configuration to the pacemaker. If the parameter values are invalid (e.g., lower rate limit exceeds the upper rate limit), it updates the communication status with an error message. If validation passes, the method sends the parameter values via serial communication and updates the status to indicate success. **Args**: None. **Returns**: None.

		Attributes Modified: self.dcm.main_page.comm_status
	send_values()	Collects the current values of all parameters in the mode by calling value for each PaceMakerParameter in self.parameters. Returns a list of these values. **Args**: None. **Returns**: list: List of current parameter values. **Attributes Modified**: None.
PaceMakerParameter	__init__(param_list, value)	Initializes a single parameter, setting up a label and slider according to param_list. Configures the slider's min, max, and initial values based on param_list, and displays the parameter's name and value in self.name_label. **Args**: param_list (list): List of parameter details (name, min, max, scaling factor). value (int): Initial value of the parameter. **Returns**: None. **Attributes Modified**: self.name, self.value, self.decimal, self.name_label, self.inputBox.
	update_value_label(value)	Updates the displayed label for the parameter when the slider is moved. This method is triggered by the slider's value change event and adjusts self.name_label to show the current value divided by self.decimal for proper scaling. **Args**: value (int): Current slider value. **Returns**: None. **Attributes Modified**: self.value.
LoginPage	__init__(dcm)	Initializes the login page, setting up the main layout, which includes the left_side (login form and buttons) and right_side (logo display). **Args**: dcm (DCM): The main DCM instance for managing application state. **Returns**: None. **Attributes Modified**: self.layout, self.left_side_layout, self.right_side_layout.
	left_side()	Sets up the left side of the login page, including a title label, login form (username and password), login and register buttons, and a warning label. Adds elements to self.left_side_layout for display. **Args**: None. **Returns**: None. **Attributes Modified**: self.username_input,

		self.password_input, self.login_warning.
	right_side()	Configures the right side of the login page with a logo, which is centered in self.right_side_layout. **Args**: None. **Returns**: None. **Attributes Modified**: None.
	handle_login()	Processes the login attempt by checking the entered username and password against self.dcm.users. If credentials match, sets self.dcm.page to 2, updates self.dcm.user, and loads user-specific pacing data into self.dcm.user_data. Calls run_gui() to proceed to the main page. **Args**: None. **Returns**: None. **Attributes Modified**: self.dcm.page, self.dcm.user, self.dcm.user_data.
	show_register_page()	Switches the application to the register page by setting self.dcm.page to 1 and calling run_gui(). **Args**: None. **Returns**: None. **Attributes Modified**: self.dcm.page.
RegisterPage	__init__(dcm)	Initializes the register page by inheriting from LoginPage and setting up additional elements in left_side() for registration-specific functionality. **Args**: dcm (DCM): Main application instance. **Returns**: None. **Attributes Modified**: Inherits and modifies elements of LoginPage.
	left_side()	Configures the left side for the registration page, including fields for username, password, and a key. Adds register and back buttons, along with a warning label for errors. Organizes these elements in self.left_side_layout. **Args**: None. **Returns**: None. **Attributes Modified**: self.reg_username_input, self.reg_password_input, self.reg_key_input, self.register_warning.
	handle_register()	Processes a registration attempt by validating the username, password, and key. If the key matches

		self.dcm.key and the username is unique, creates the user in users.txt by calling write_user(). If registration succeeds, clears inputs and displays a success message. **Args**: None. **Returns**: None. **Attributes Modified**: None, though it updates self.dcm.users by calling write_user().
	show_login_page()	Returns to the login page by setting self.dcm.page to 0 and calling run_gui(). **Args**: None. **Returns**: None. **Attributes Modified**: self.dcm.page.
LiveGraphWidget	__init__(dcm, length=10, frequency=100)	Initializes the live graph widget for visualizing atrial and ventricular data in real-time. Sets up the plotting environment, initializes data buffers, and configures the user interface with tabs for different visualizations. **Args:** dcm (DCM): Instance of the DCM class providing access to the pacemaker's data and state. length (int, optional): Duration of the graph in seconds. Defaults to 10. frequency (int, optional): Update frequency in Hz. Defaults to 100. **Returns:** None. **Attributes Modified:** self.dcm self.length self.frequency self.y_min self.y_max self.atrial_data self.ventricle_data self.time_data self.port_device self.timer_interval
	init_ui()	Sets up the user interface for the live graph widget. Creates tabs for visualizing atrial, ventricular, and combined data, configures matplotlib plots, and adds buttons for starting, pausing, and saving the graph. **Args:** None

		Returns: None **Attributes Modified:** self.tabs self.tab_1 self.tab_2 self.tab_3 self.figure_1 self.ax_1 self.canvas_1 self.figure_2 self.ax_2 self.canvas_2 self.figure_3 self.ax_3 self.canvas_3 self.start_button self.pause_button self.save_button self.timer
	update_plot()	Updates the graph with new atrial and ventricular data retrieved via serial communication. Adjusts the X and Y axes dynamically to accommodate the latest data and redraws the plot. **Args:** None **Returns:** None **Attributes Modified:** self.time_data self.atrial_data self.ventricle_data self.atrial_line_1 self.ventrical_line_2 self.atrial_line_3 self.ventrical_line_3 self.y_min self.y_max
	start_live_plot()	Begins the real-time plotting by setting up a timer to call update_plot() at regular intervals. Initializes the serial connection and resets the live plot. **Args:** None **Returns:** None **Attributes Modified:** self.port_device self.start_time self.timer

	pause_live_plot()	Pauses the real-time plotting by stopping the update timer. Updates the communication status to indicate that data transfer has stopped. **Args:** None **Returns:** None **Attributes Modified:** self.timer
	reset_live_plot()	Stops the live plot and resets all data buffers and plot axes to their initial states. Clears existing plot lines and redraws the canvas. **Args:** None **Returns:** None **Attributes Modified:** self.y_min self.y_max self.time_data self.atrial_data self.ventricle_data
	save_graph	Saves the current state of the graphs to PNG files. Each tab's graph (atrial, ventricular, and combined) is saved as a separate file. **Args:** None **Returns:** None **Attributes Modified:** None
Stylemanager	__init__()	Initializes the StyleManager instance, setting the default theme to light and generating the initial stylesheet. **Args:** None **Returns:** None **Attributes Modified:** self.themes self.current_theme self.stylesheet
	set_theme(theme_name)	Changes the current theme to the specified theme name, if valid, and updates the stylesheet accordingly.

		Args: theme_name (str): The name of the theme to apply. Must be one of the keys in THEMES. **Returns:** None **Attributes Modified:** self.current_theme self.stylesheet
	update_styles()	Generates a dynamic stylesheet based on the currently selected theme. This method formats the style rules for PyQt widgets such as QMainWindow, QPushButton, QLabel, QLineEdit, QSlider, QTabWidget, and QScrollBar. **Args:** None **Returns:** None **Attributes Modified:** self.stylesheet
SerialComs	check_connection(ser_num)	Checks for a serial connection to a device with the specified serial number. Searches through available COM ports, matches the serial number in the hardware ID, and returns the corresponding port name. **Args:** ser_num (str): The serial number of the device to search for. **Returns:** str: The name of the COM port connected to the device if found (e.g., "COM3"). None: If no matching device is found or no COM ports are available. **Attributes Modified:** None.
	serial_comm(port_name, sync_in, func_in, parameters=None, mode=16)	Performs serial communication with a connected device via the specified COM port. Sends data to the device, including operational parameters, and optionally receives a response. This function formats the data into a byte structure and writes it to the serial port. It supports functionality for sending pacing parameters or other data to a pacemaker device. **Args:** port_name (str): The name of the COM port to communicate with (e.g., "COM3"). sync_in (int): Synchronization input value to include in the data packet.

		func_in (int): Function code to include in the data packet (e.g., 85 for sending data, 34 for reading data). parameters (dict, optional): A dictionary of parameter values to override defaults. Defaults to None. mode (int, optional): The mode to set for the device. Defaults to 16. **Returns:** tuple: A tuple containing the unpacked response from the device (if applicable) None: If there is no response or an error occurs during communication. **Attributes Modified:** None.

The programmable parameters in the DCM originate from the user interface, where doctors input values using sliders for each pacing mode. These values are captured and organized in a dictionary structure, with mode names as keys and parameter lists as values. Default parameters are initialized during the DCM setup, and user-modified values are stored in a text file to ensure persistence across sessions. Parameters are transmitted to the pacemaker via serial communication, formatted as commands with mode names and corresponding parameter values. To ensure the integrity of parameters, the DCM employs a two-way communication process, verifying that parameters stored in the pacemaker match the values input by the doctor. This involves a parameter readback mechanism, checksum validation during transmission, and real-time status indicators for communication and device health.

To optimize communication, parameters such as the Atrial Amplitude are transmitted as scaled integers. For example, an Atrial Amplitude of 0.1 V is scaled as 1 during transmission, fitting within the communication constraints, the range of data sent must be between 0 and 255. The pacemaker through Simulink reverses this scaling by dividing the received integer by the scaling factor, ensuring that the parameters are correctly interpreted and implemented in the device. This consistent scaling and descaling mechanism ensures the values input by the doctor are preserved and accurately applied to the pacemaker.

The choice of integers as data types simplifies the communication protocol and ensures compatibility with the structured binary format required by the serial interface. The dictionary structure used within the DCM supports efficient organization and quick lookups of mode-specific parameters, while the text file storage maintains simplicity and cross-platform usability. These combined measures ensure a robust and accurate parameter handling process from the DCM to the pacemaker.

Testing Results - DCM

Registration - Case 1:

The doctor key is 1234. With no usernames in place at the moment, an attempt was made to make an account with the username john and password snow. However, the key is clearly incorrect.

Table 9: Incorrect key

Case	Input	Expected Output	Output
New username, incorrect key	Username: john Password: snow Key: 123	Incorrect Key	Incorrect Key

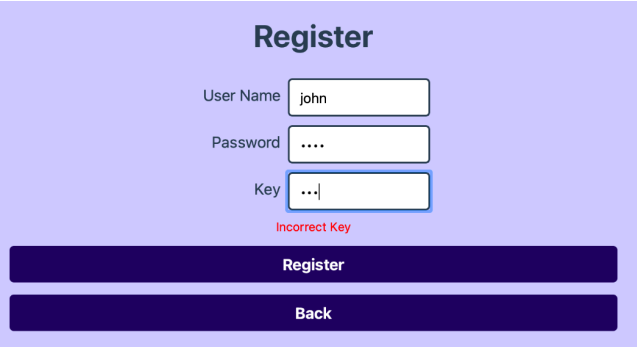

Figure 42: Registration Case 1

Registration - Case 2:

Once the key was changed to 1234, the registration was successful.

Table 10: Correct key

Case	Input	Expected Output	Output
New username, correct key	Username: john Password: snow Key: 1234	Registration successful	Registration successful

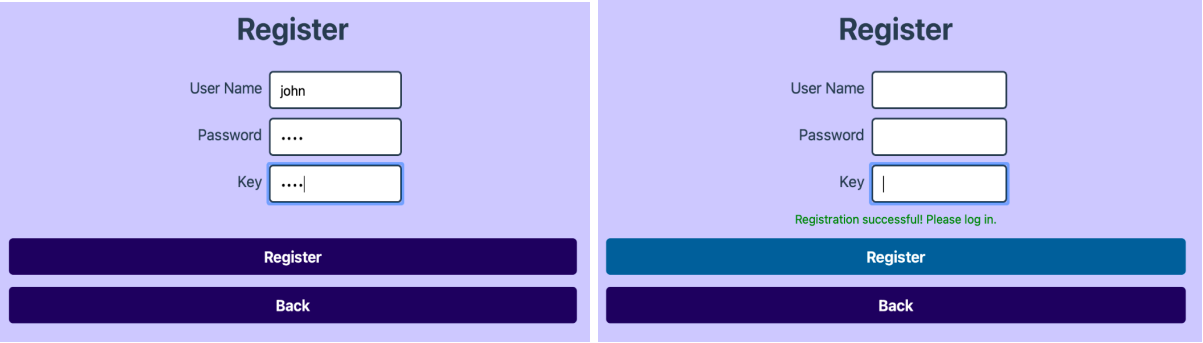

Figure 43: Registration Case 2

Registration - Case 3:

Attempting to once again make another account, this time with the correct key and an existing username, the error clearly states the username exists, indicating registration was unsuccessful.

Table 11: Existing username

Case	Input	Expected Output	Output
Existing username, correct key	Username: john Password: snow Key: 1234	Username Already Exists	Username Already Exists

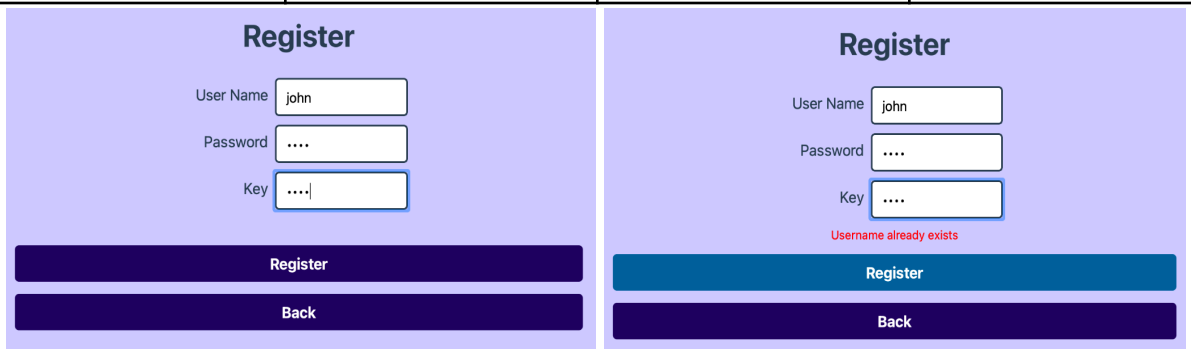

Figure 44: Registration Case 3

Registration - Case 4:

Attempting to make another account, this time with the incorrect key and an existing username, the error clearly states the key is incorrect, since the incorrect doctor key is a more pressing matter than the username existing.

Table 12: Incorrect key & existing username

Case	Input	Expected Output	Output

Existing username, incorrect key	Username: john Password: snow Key: 123	Incorrect Key	Incorrect Key

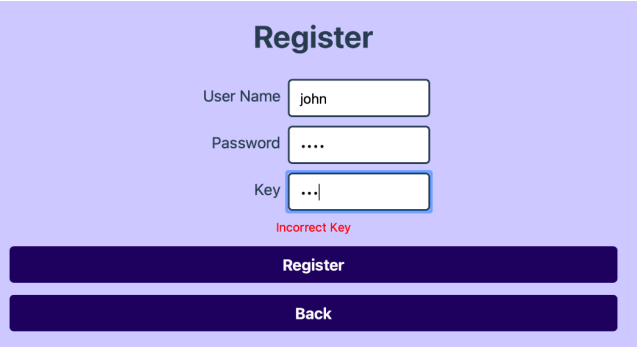

Figure 45: Registration Case 4

*It is important to note that any Password will be taken

Registration - Case 5:

Attempting to make another account, this time with the correct key and a new username, while having reached the maximum of 10 users. The returned error is Maximum Users Reached

Table 13: Maximum users

Case	Input	Expected Output	Output
New username Correct key 11th Account	Username: john10 Password: snow Key: 1234	Maximum Users Reached	Maximum Users Reached

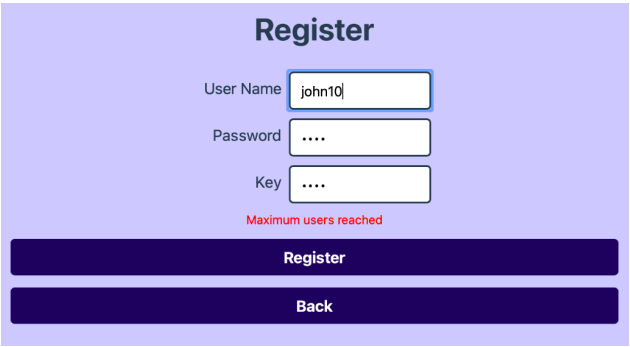

Figure 46: Registration Case 5

Login - Case 1:

Attempting to login with a correct username and incorrect password, the error warns the username or password is incorrect and the login is unsuccessful.

Table 14: Incorrect password

Case	Input	Expected Output	Output
Existing username, incorrect password	Username: john Password: snowwyyy	Incorrect Username or Password	Incorrect Username or Password

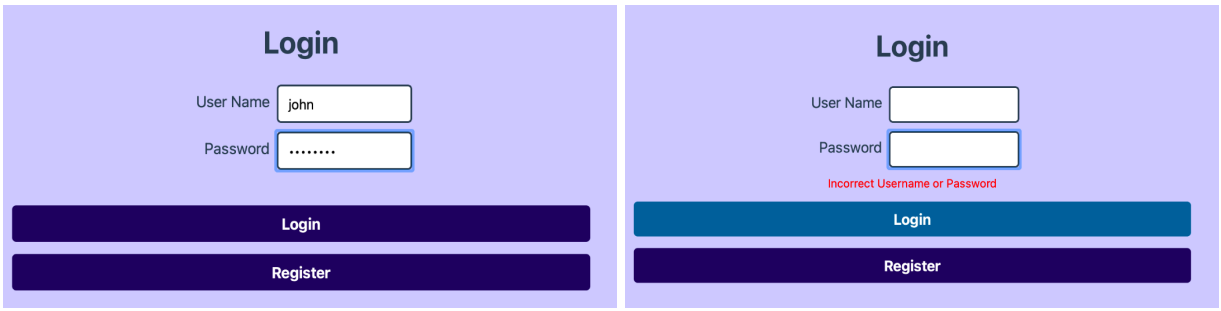

Figure 47: Login Case 1

Login - Case 2:

Attempting to login with an incorrect username and existing password from the locally stored file, the error warns the user that the username or password is incorrect and the login is unsuccessful.

Table 15: Incorrect username

Case	Input	Expected Output	Output
Incorrect username, Existing password	Username: jonny Password: snow	Incorrect Username or Password	Incorrect Username or Password

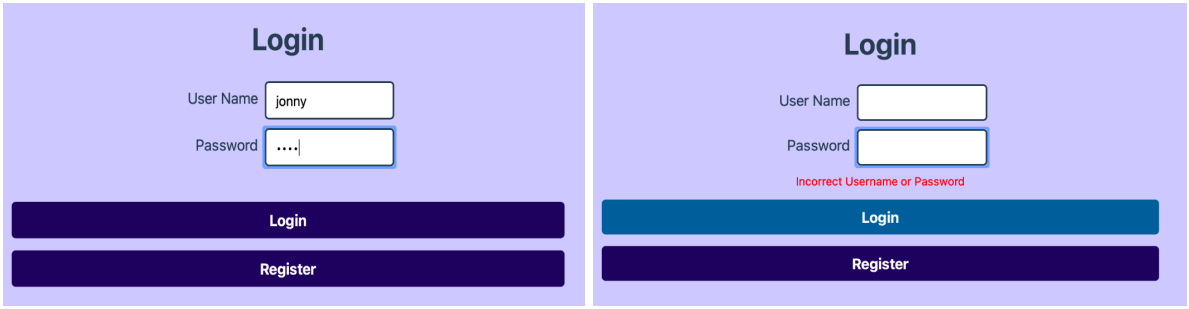

Figure 48: Login Case 2

Login - Case 3:

Attempting to login with an incorrect username and password that does not exist in the locally stored file, the error warns that the username or password is incorrect and the login is unsuccessful

Table 16: Maximum users

Case	Input	Expected Output	Output
Incorrect username, incorrect password	Username: jonny Password: snowwyyy	Incorrect Username or Password	Incorrect Username or Password

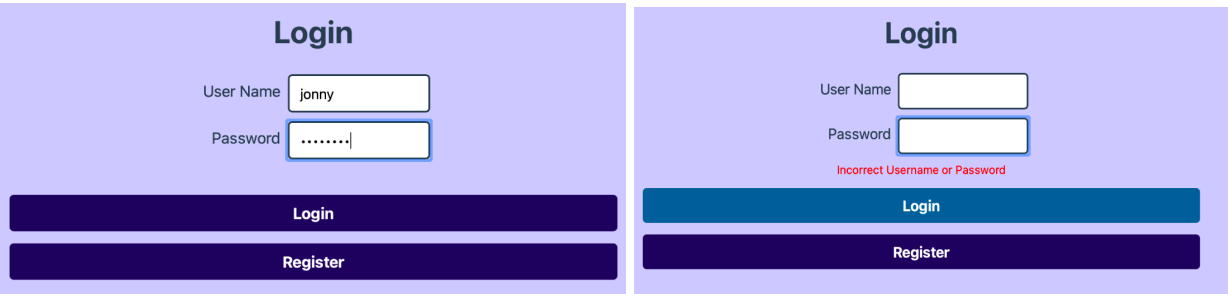

Figure 49: Login Case 3

Login - Case 4:

Attempting to login with a correct username and password, the screen goes to the mainpage

Table 17: Maximum users

Case	Input	Expected Output	Output
Incorrect username, incorrect password	Username: john Password: snow	Switch to mainpage	Switch to mainpage

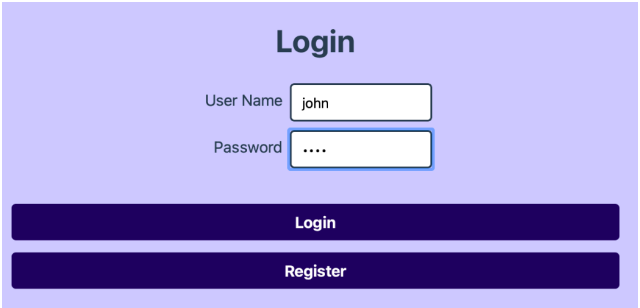

Figure 50: Login Case 4

Main Page Test Cases:

Sign Out Button:

Initial:

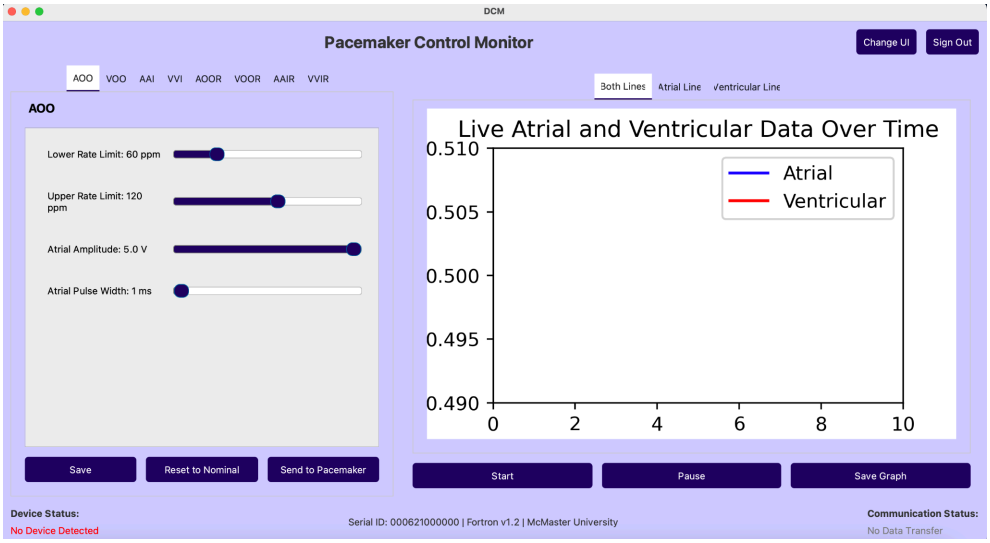

Figure 51: DCM Interface Before Clicking SIgnout

After:

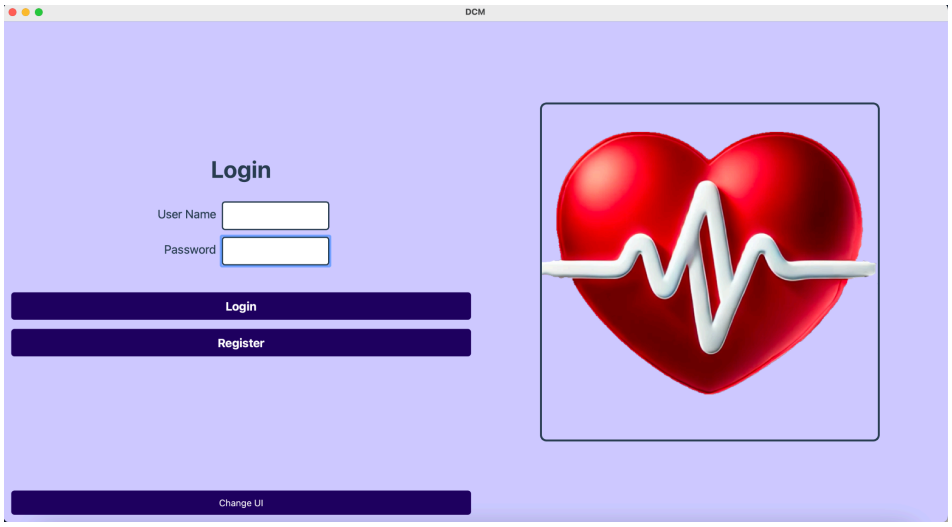

Figure 52: DCM Interface After Clicking SIgnout

Checking Saved Parameters When Switching Modes:

Initial:

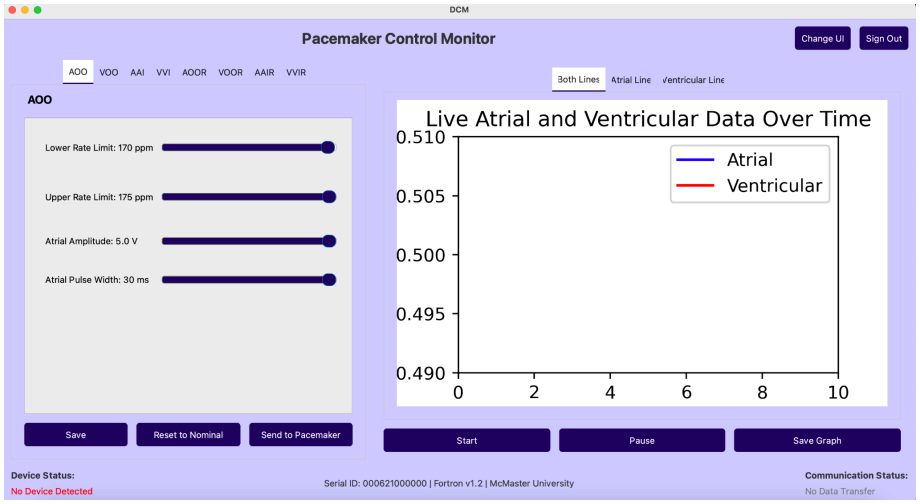

Figure 53: DCM While Setting Up Mode AOO Parameters

Switching:

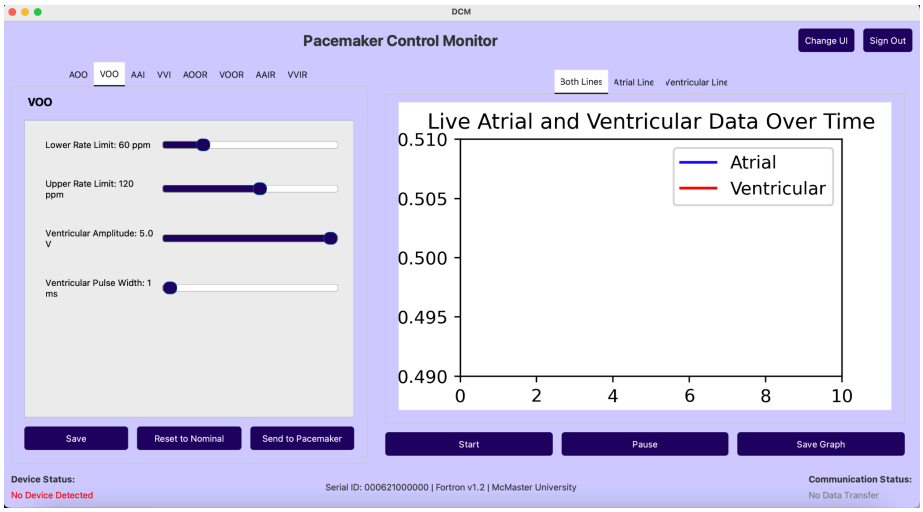

Figure 54: DCM While Setting Up Mode VOO Parameters

After:

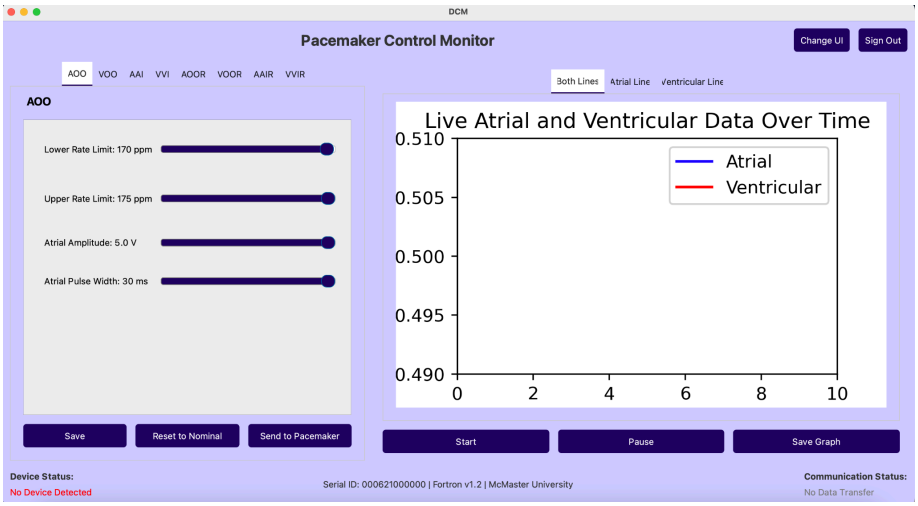

Figure 55: DCM While Setting Up Mode AOO Parameters. The sliders and the values stayed the same as expected.

Checking Saved Parameters When signing out and signing back in:

Initial:

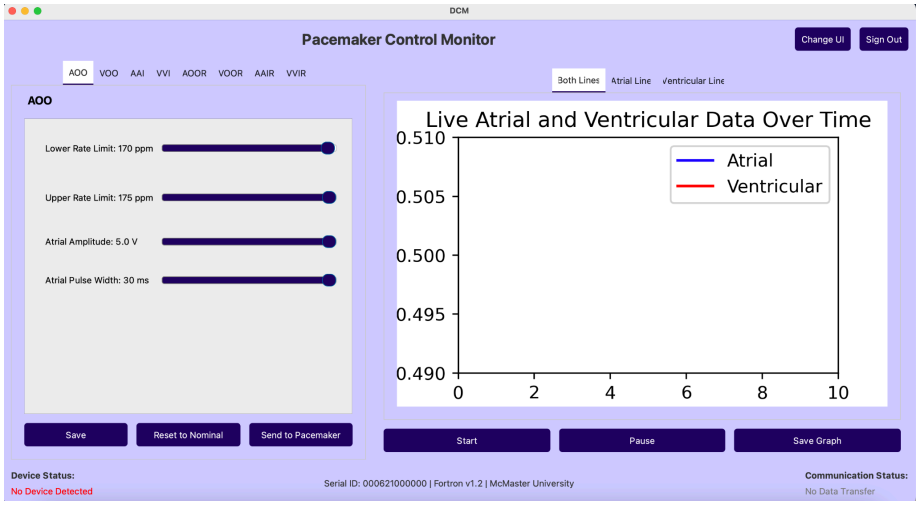

Figure 56: DCM While Setting Up Mode AOO Parameters

*Save is Pressed Here

Switching:

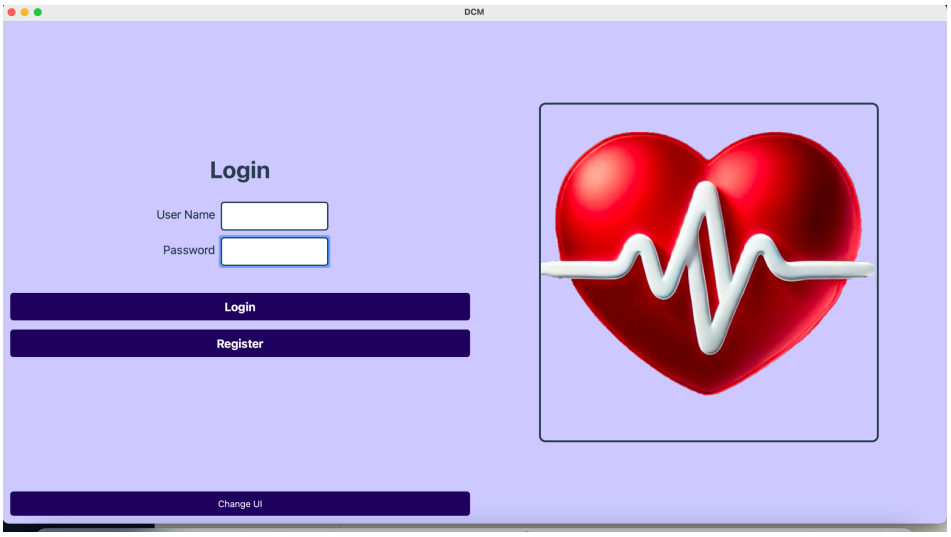

Figure 57: DCM While Signed Out

*Logged In with same User

After:

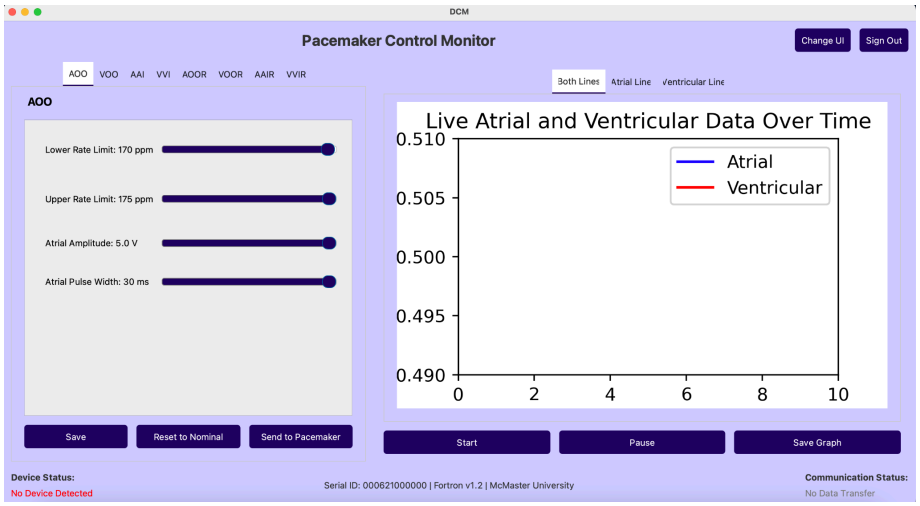

Figure 58: DCM While Logged In

*Display comes Like this

Checking Reset to Nominal Button:

Initial:

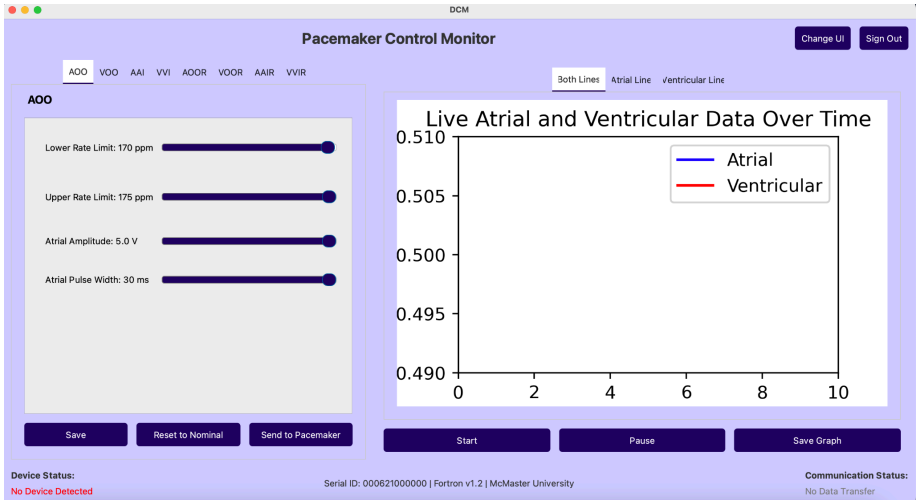

Figure 59: DCM While Setting Up Mode AOO Parameters

*Reset to nominal button clicked

Switching:

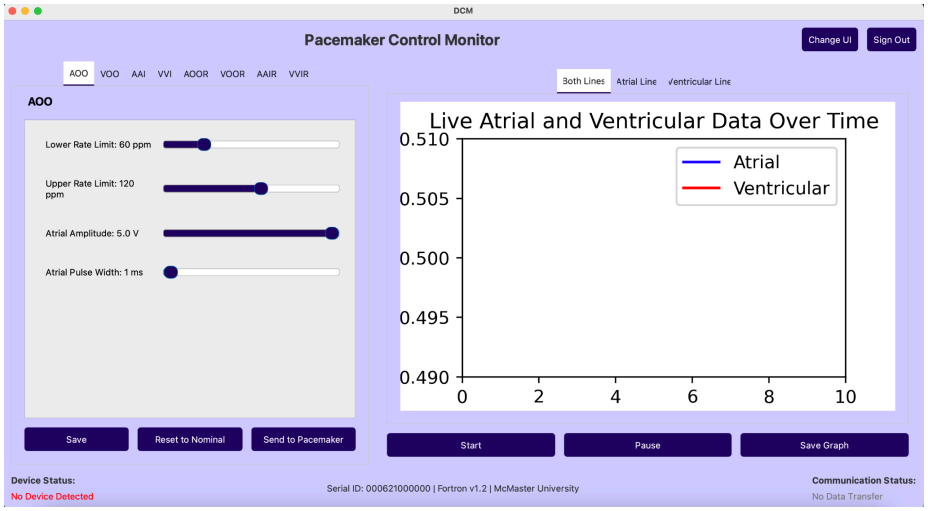

Figure 60: DCM While Setting Up Mode VOO Parameters

After:

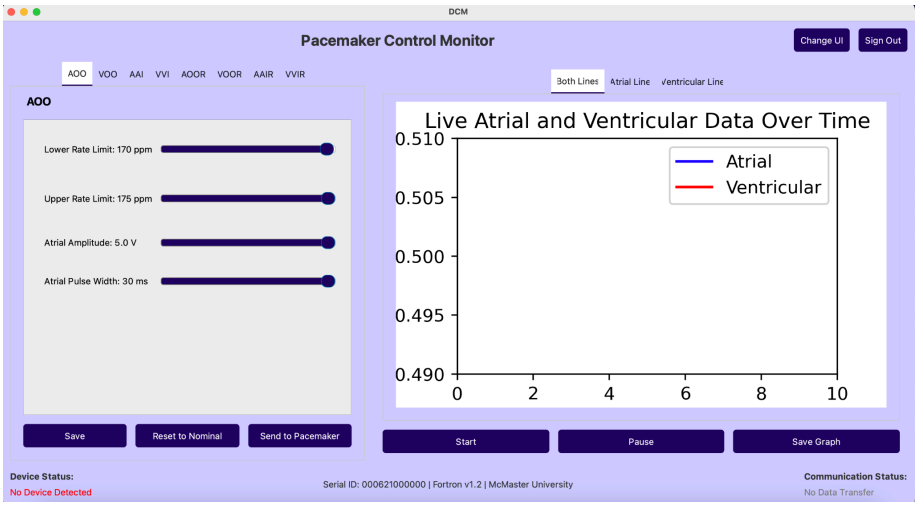

Figure 61: DCM While Setting Up Mode AOO Parameters. The sliders and the values stayed the same as expected.

Checking Change Ui Button:

Default:

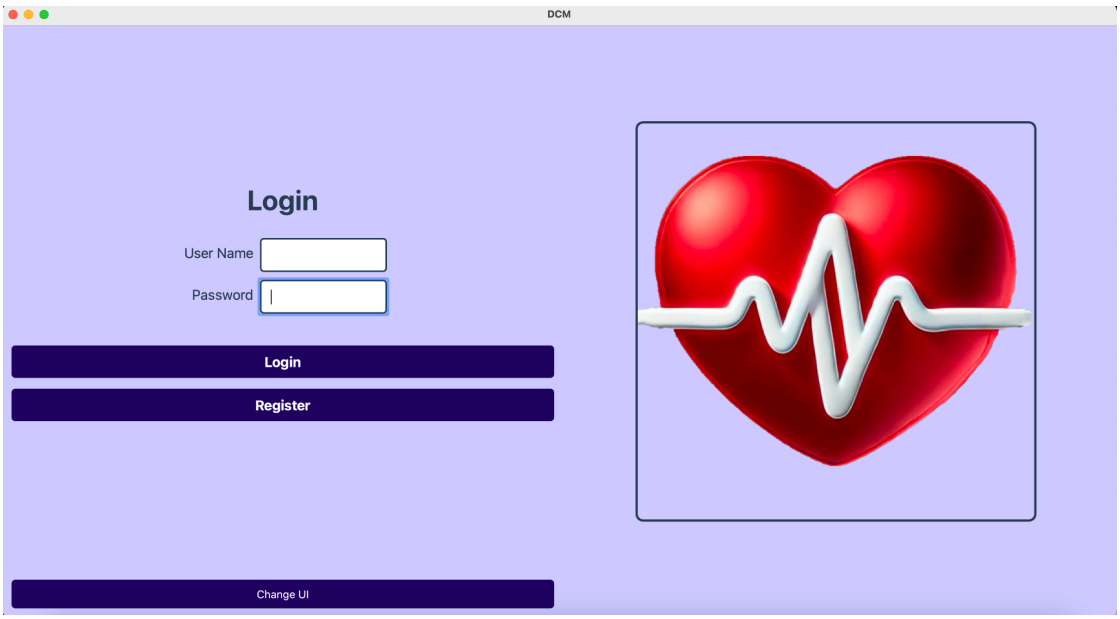

Figure 62:

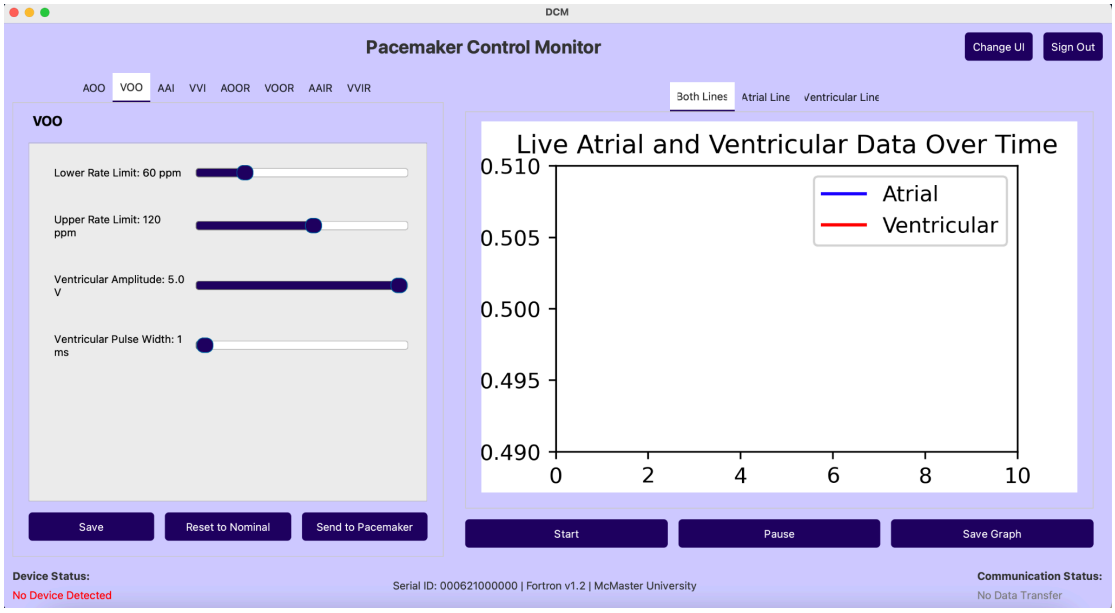

Figure 63:

Colour 2:

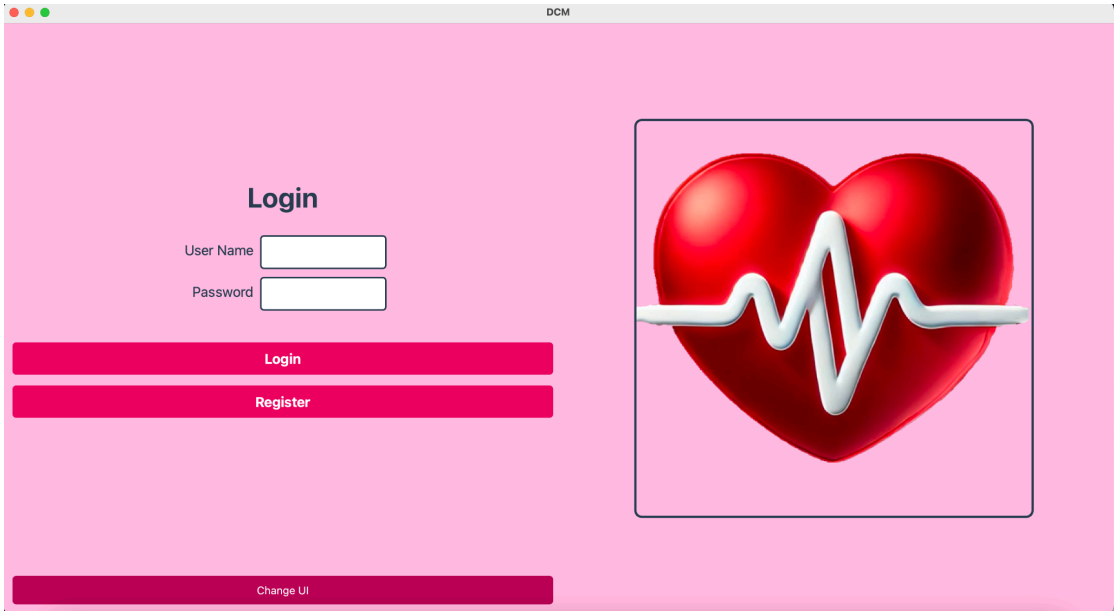

Figure 64:

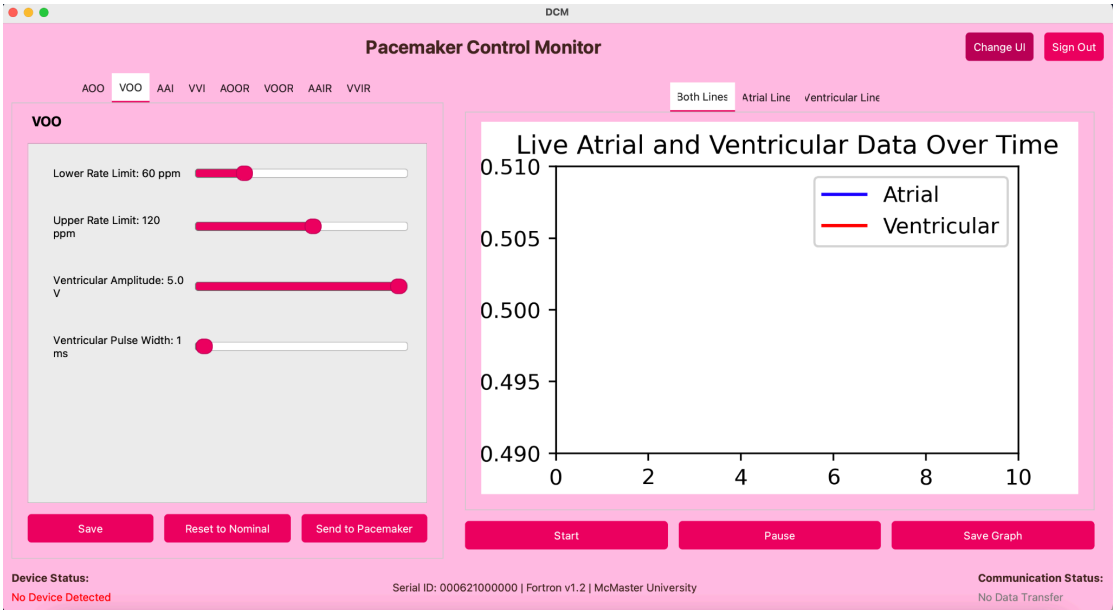

Figure 65:

Colour 3:

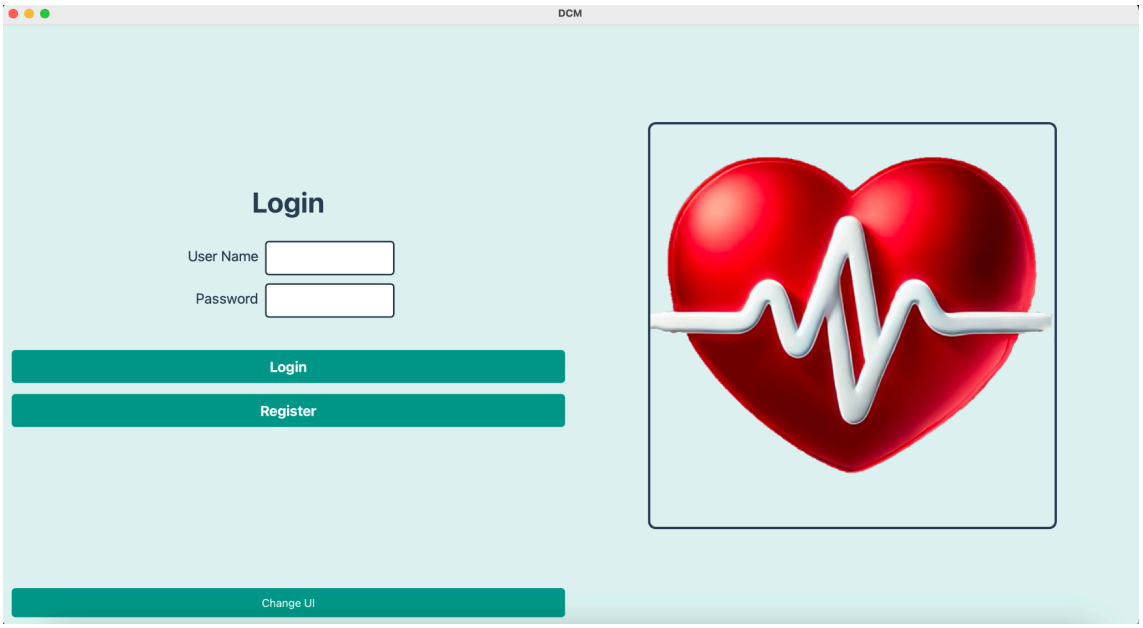

Figure 66:

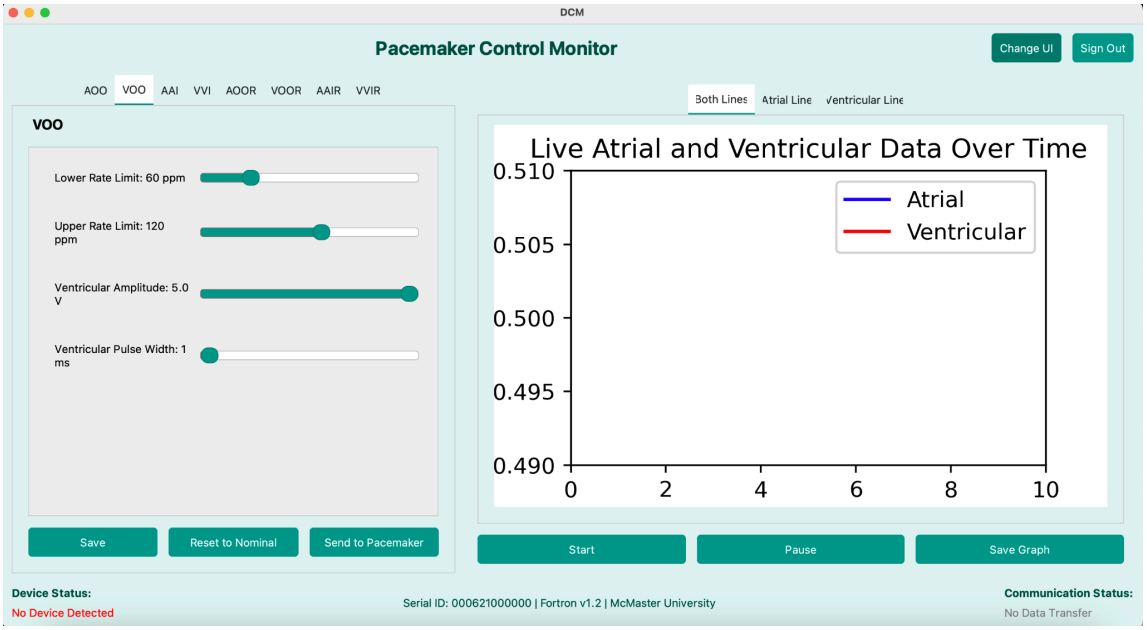

Figure 67:

Checking Device Status:

Before connecting:

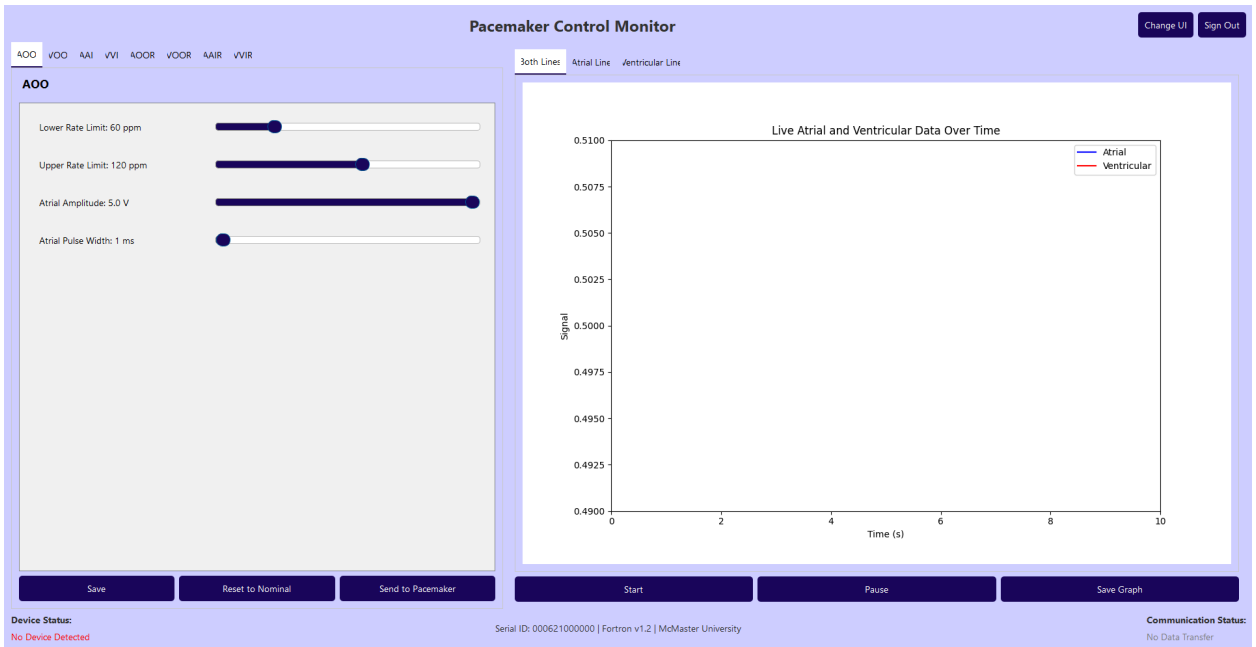

Figure 68: No device connected

After connecting:

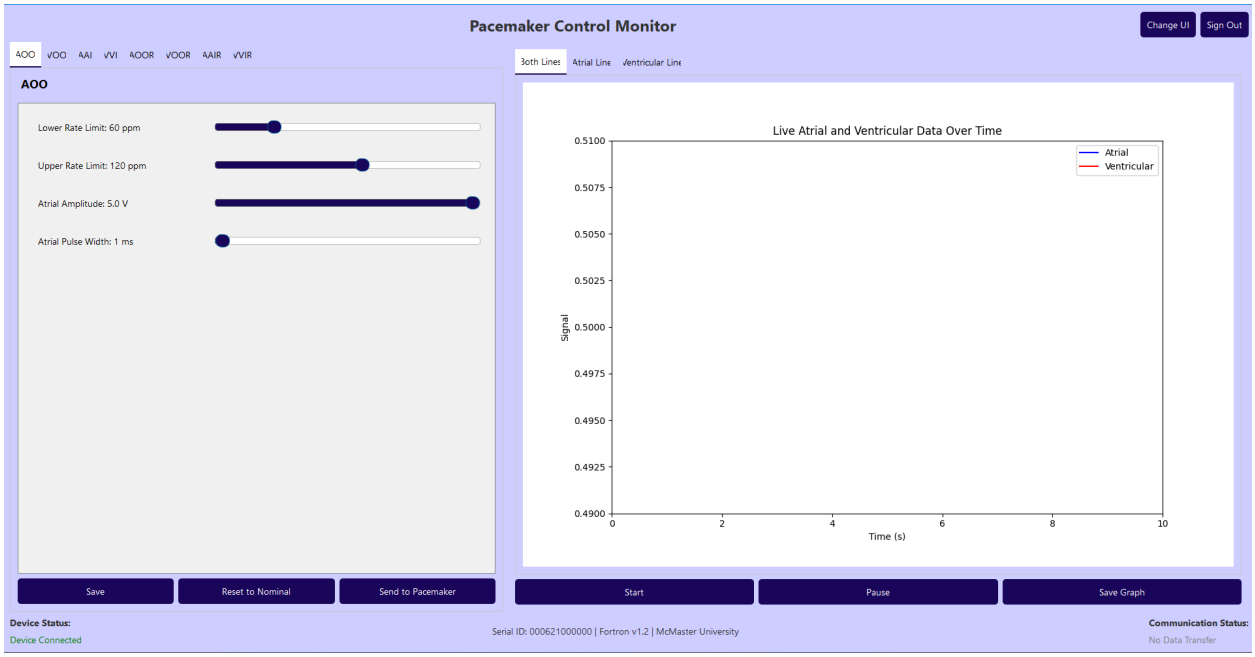

Figure 69: Device Connected

Checking Send to Pacemaker Button/Communication Status:

Correct Values:

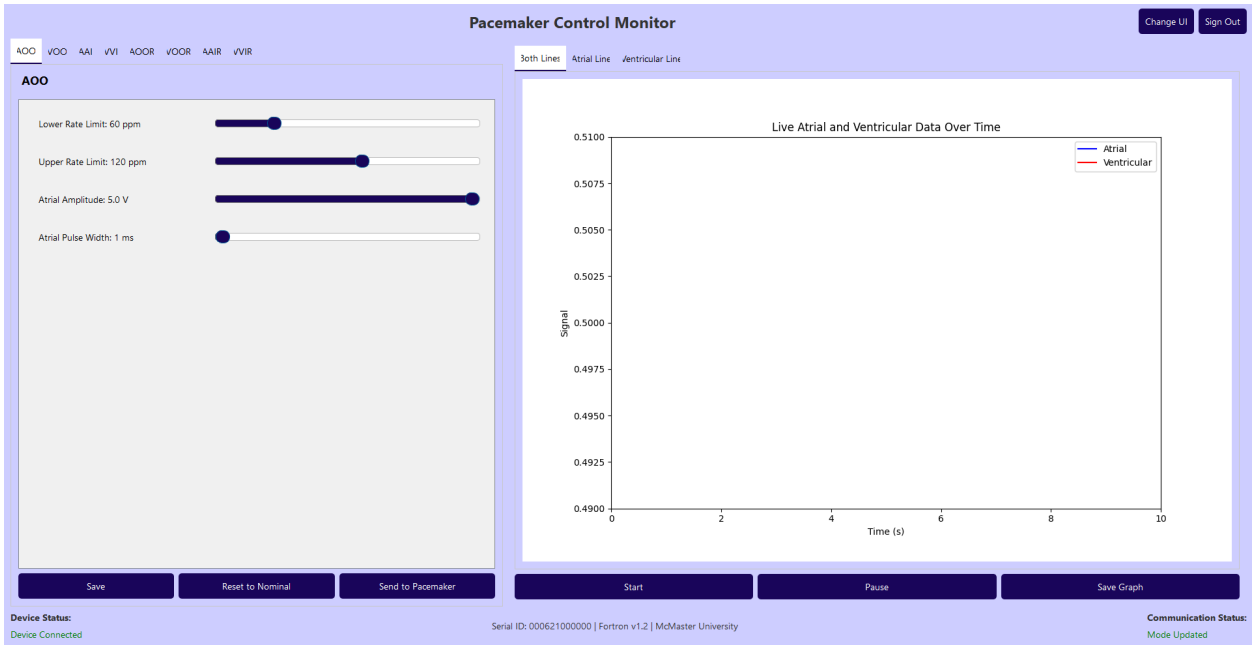

Figure 70: Serial Communication working

Incorrect values (Lower rate limit greater than upper rate limit or maximum sensor rate not in between the two):

Figure 71: Invalid parameters shown, Maximum sensor rate greater than upper rate limit

Checking Graph:

Before:

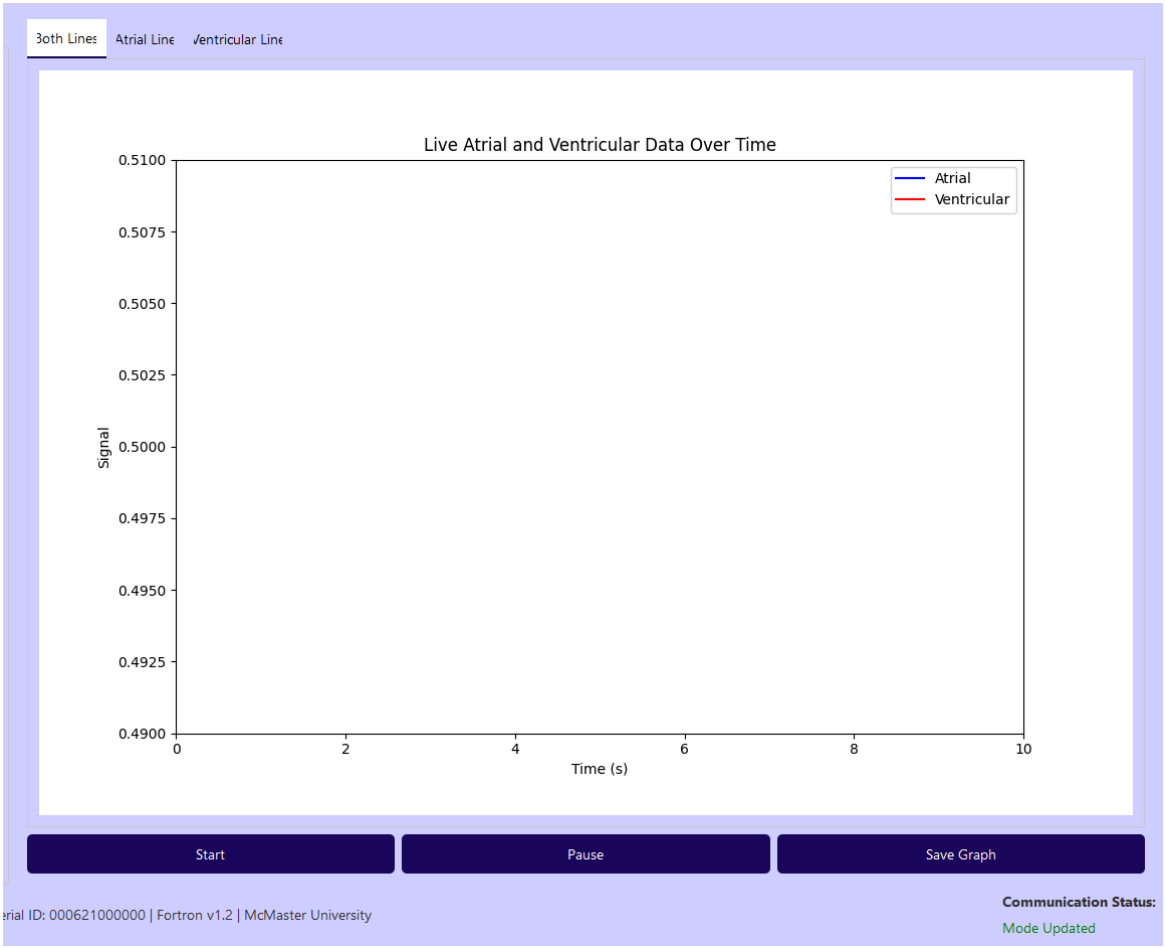

Figure 72: Graph before starting

Start:

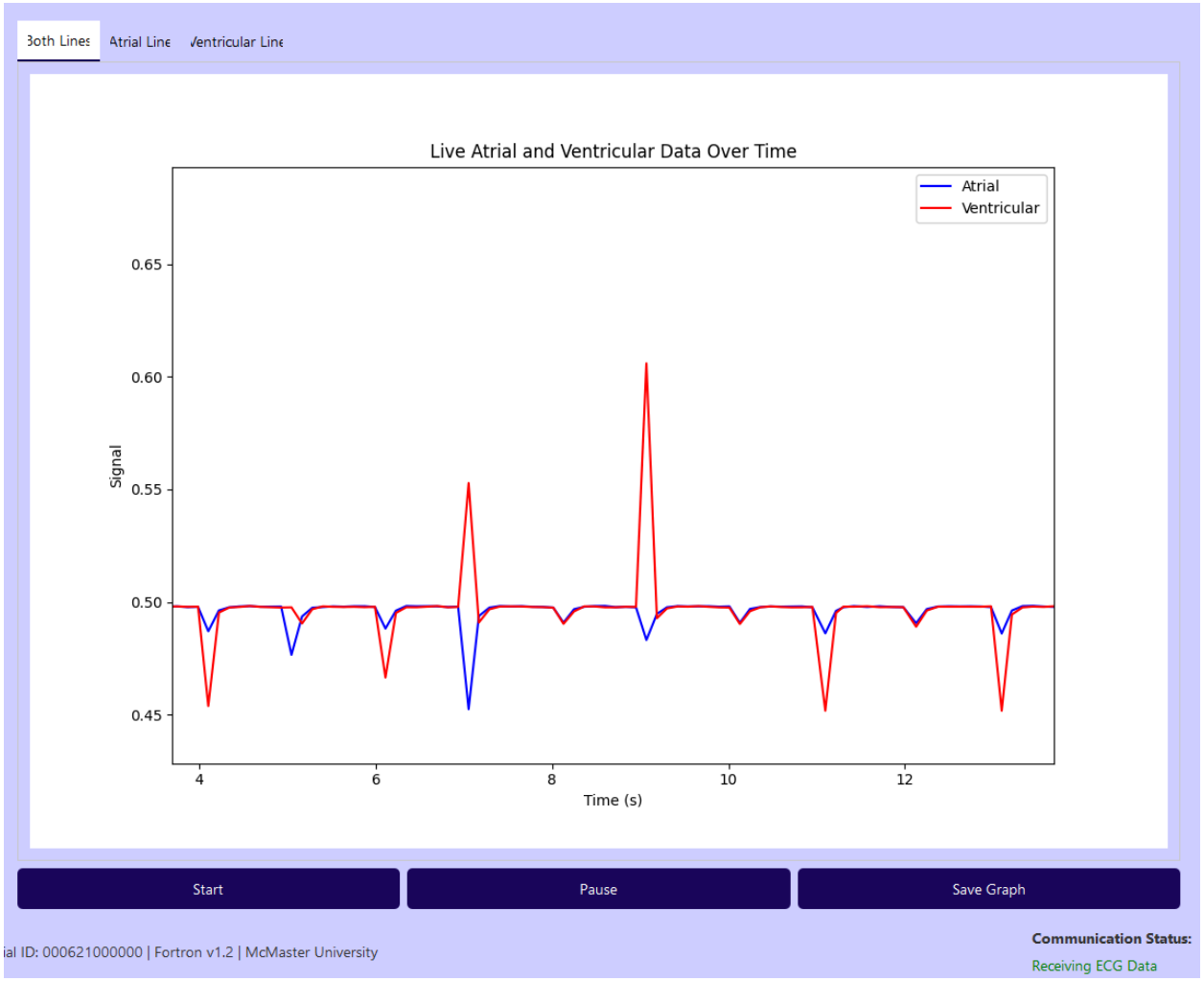

Figure 73: Starting functionality

Pause:

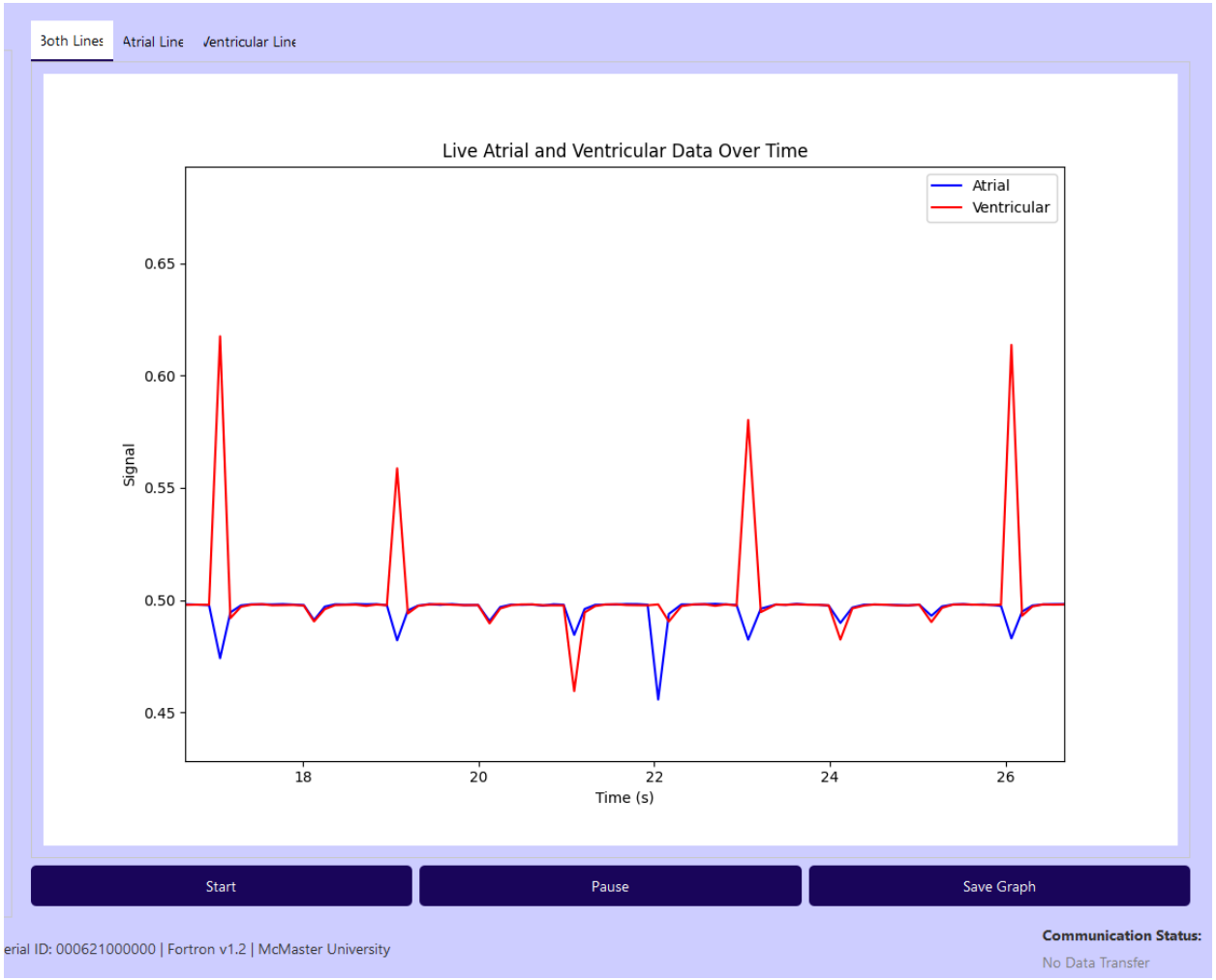

Figure 74: Pause functionality

Save Graph

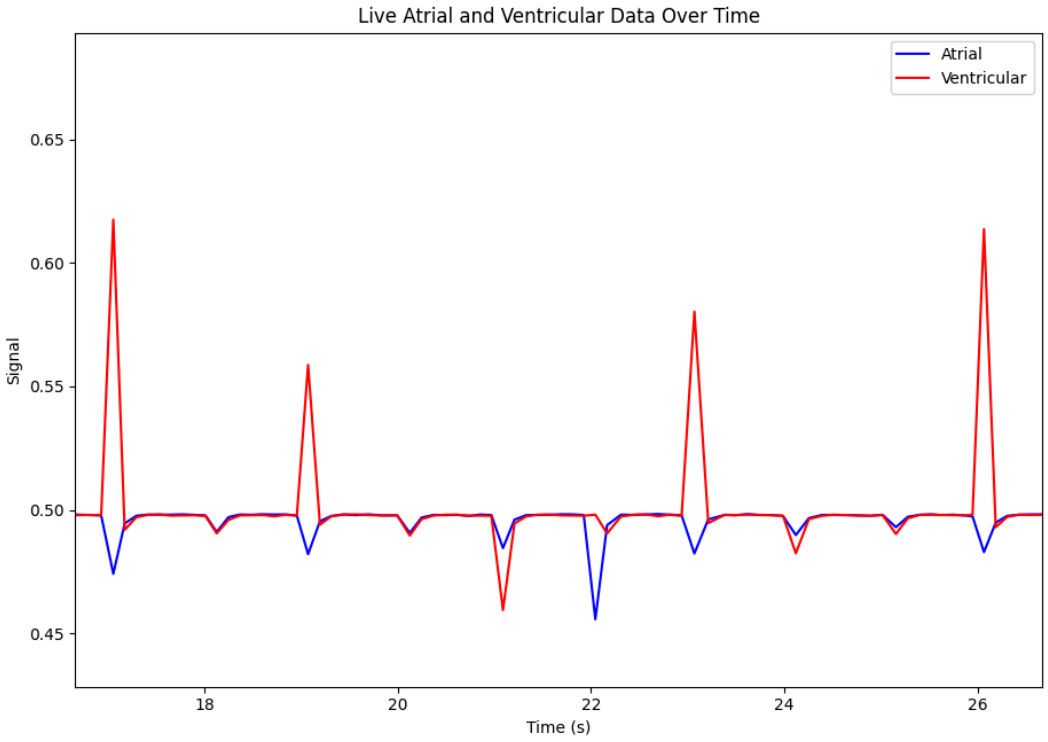

Figure 75: Both plots over time

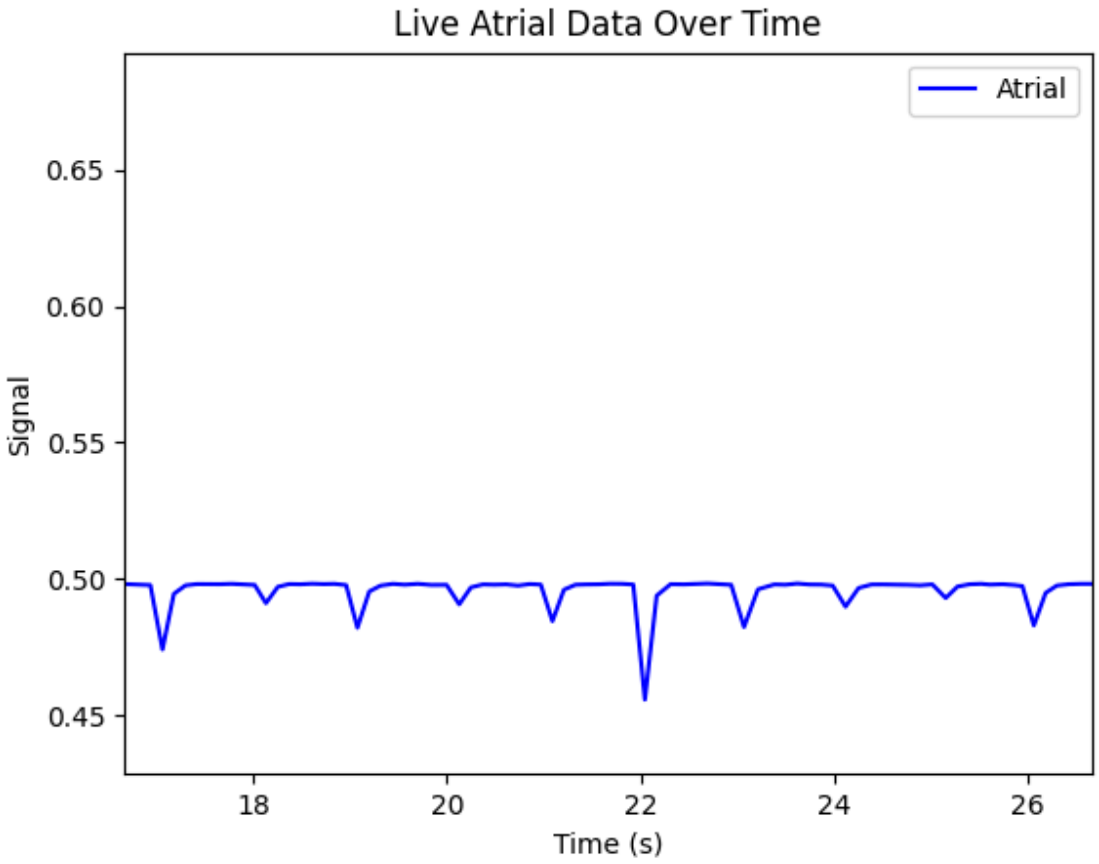

Figure 76: Atrial Data

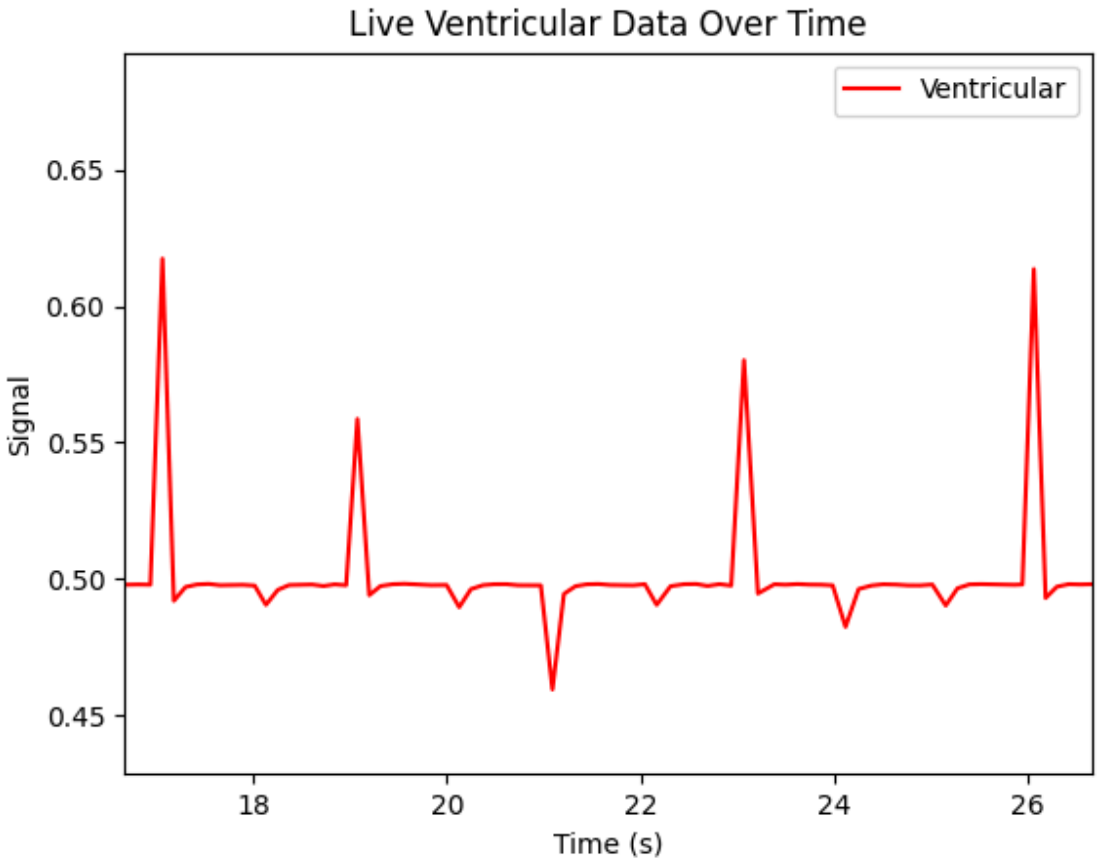

Figure 77: Ventricular Data

Problems and Challenges:

DCM

Assignment 1

The initial challenge of creating the DCM didn't seem as daunting. Once the codebase became extremely large, it was harder to navigate. Initially, the basic code was developed in one file. It was intended that the code would reflect the OOP/Modularity standards that we were taught in class while also being much easier to debug. Once the basic functionality of the code was completed, it was decided that the code would be split into files with related methods and classes, so specific parts of the user interface are easier to find and change. Initially, the only extra file created was the parameter display file which was a great display of efficient and object-oriented code. In an attempt to further the modularity, every page in the ui was made its own OOP. There were 3 pages, aside from the 4 modes which were already encompassed in the parameter display function. The login and registration pages were quite similar in look and placement. So the registration page was made a child class of the login page class. In addition, the registration class had a method overwritten over the parent class login page. Since it was decided that the next page would be the main UI page, it would mean that the actual values of the parameters were hidden under 4 layers of objects (DCM, MainPage, Pace Maker Mode, and Pace Maker Parameter, at least when initializing). This became a tedious task when it was needed for the saved values to be loaded back into the pacemaker.

Another problem faced was the existence of layouts in pyQt. Initially, in the lesser OOP version of the code, layouts were returned whenever run_gui() was called. After the changes, it just creates a class of the layout which was called upon each time. When the layout was changed to another page, the initial layout would be deleted. This means that the original address would be written over so that when the code would return to the previous layout object, it would cause an error and crash the program. Debugging was difficult, and no information researched was useful. The solution was found accidentally when the objects were defined again in the run_gui() class.

Overall the most challenging task was setting up the database to receive and write the username, password and data. Then sending the data all the way into the parameter values.

Assignment 2

One of the challenges faced was getting the serial communication to work. It was difficult attempting to implement UART communication through the DCM from the beginning. Due to the major challenges that arose implementing this and the fact that UI and serial communication changes were concurrently in progress, an alternative development process was needed. Since the UART and egram graphing changes had to work, the solution was that they would be implemented in files independent of the DCM, while the UI changes were made to account for this - separately. One of the major downsides to PaceMaker was that it didn't have anything like print statements, so it wasn't easy to troubleshoot

with it. Once the transmit portion of serial communication was working, it was close to impossible to test it. We had implemented some flashing lights to fix this issue. We realized much later that our transmit functions were working and the integration was good, but only after creating the receive function on the DCM end. This problem arose from the fact that the DCM team relied on the Simulink code and the DCM team wasn't well versed with Simulink code. Once serial communications were working, while the code to graph was quite lengthy, it wasn't much of a challenge.

Simulink

Assignment 1

Several issues were encountered in developing the Simulink model, particularly with the hardware. One significant issue was that the Simulink model worked on one of the FRDM-K64F boards but not the other. The issue ended up involving the incorrect setting on the switch. It was set to 1x when it should have been set to 100x. Theoretically, the pacemaker should pulse below the lower rate limit and above the upper rate limit. This is to address biological dysfunctions properly. However, having the switch set to 1x made it so that the pacemaker would not pulse for any heartbeat rate.

Additionally, we discovered that the output block for pins D6 and D3 was set to a digital write block when it should have been a PWM output block. This impacted the output signals and as a result, our sensing circuit. Lastly, we discovered that the front-end control block (D13) was missing, meaning that the sensing circuitry was never activated, thus inhibiting accurate heart signal detection. Overall, we understood the logic of this project, however, translating it into actual implementation proved to be complex.

Assignment 2

During Assignment 2, challenges arose for Rate Adaptive mode logic and serial communication. In the Rate Adaptive Subsystem MATLAB function block, The if loop testing whether Activity_Level is less than -10 was created after testing Rate Adaptive modes. The accelerometer was not responding when shaking the pacemaker even after a test in another simplified Simulink model confirmed that it was functional. Therefore, it was hypothesized that the code logic was overpenalizing very low activity levels when taking the difference of the threshold and the activity level. This meant that the calculated acceleration was too low and did not visually increase the desired pace rate when being added to the previous rate. Once this loop was implemented to establish the minimum activity level, all rate adaptive modes worked as expected and 300 samples could be maintained in the Moving Average block.

A big issue also was making sure all data types were consistent across the model. Simulink operations like muxing and byte packing require uniform data types. Any mismatch, like mixing double and single, caused errors. To resolve this, we explicitly converted variables, validated the model, and debugged for compatibility throughout the system.

Next, setting up serial communication with the DCM was complex due to the need for byte management and transmission order. Each data packet had to have the correct number of bytes, and the order of parameters and signals had to match the DCM's expectations. We used byte-packing blocks to serialize data, but this required analyzing the configuration and multiple iterations to ensure the data was transmitted correctly and interpreted reliably by the DCM.

Testing Results - Pacemaker

Testing the Pacemaker modes was done by changing the MODE input parameter to the number corresponding to the mode, and then using heartview to visualize the pacemaker's pacing and sensing properties.

The following test cases demonstrate the pacemaker's pulsing and sensing functionality:

AOO - Case 1

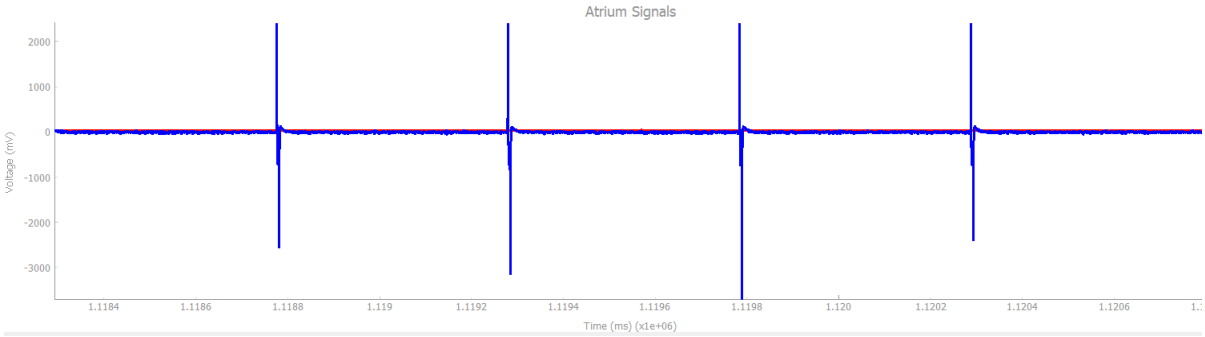

Figure 78: AOO Test Case 1 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = OFF

AOO - Case 2

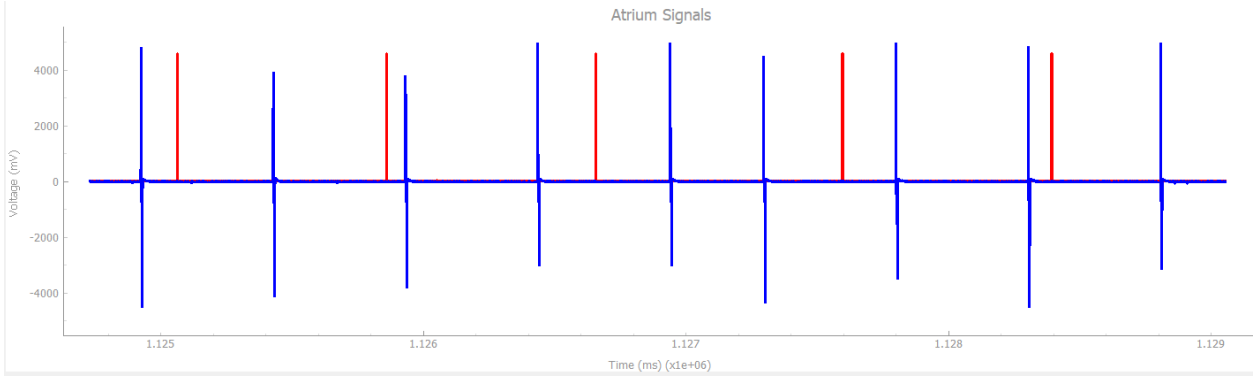

Figure 79: AOO Test Case 2 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = ON,

BPM = 75

VOO - Case 1

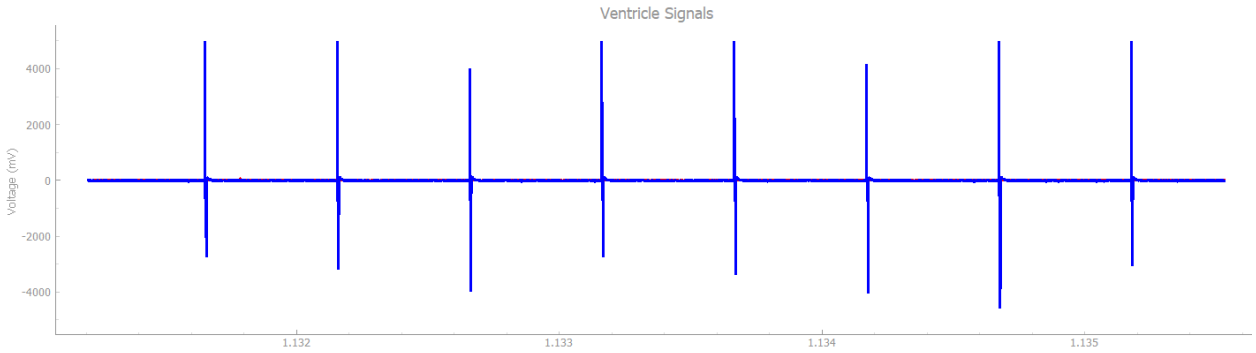

Figure 80: VOO Test Case 1 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = OFF

VOO - Case 2

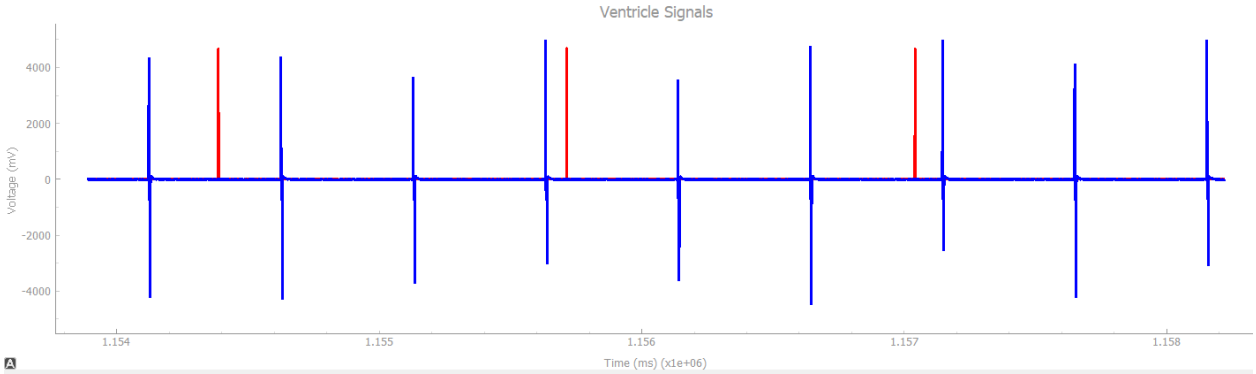

Figure 81: VOO Test Case 2 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 45

AAI - Case 1

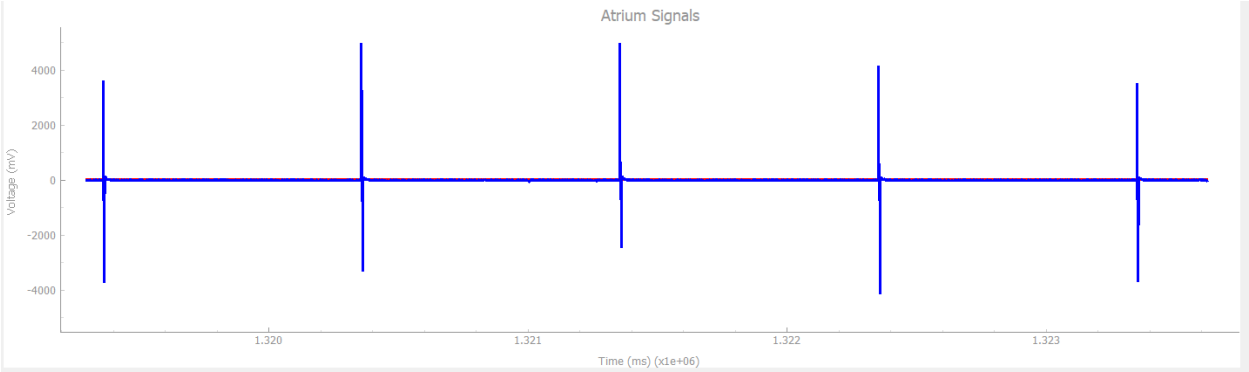

Figure 82: AAI Test Case 1 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = OFF

AAI - Case 2

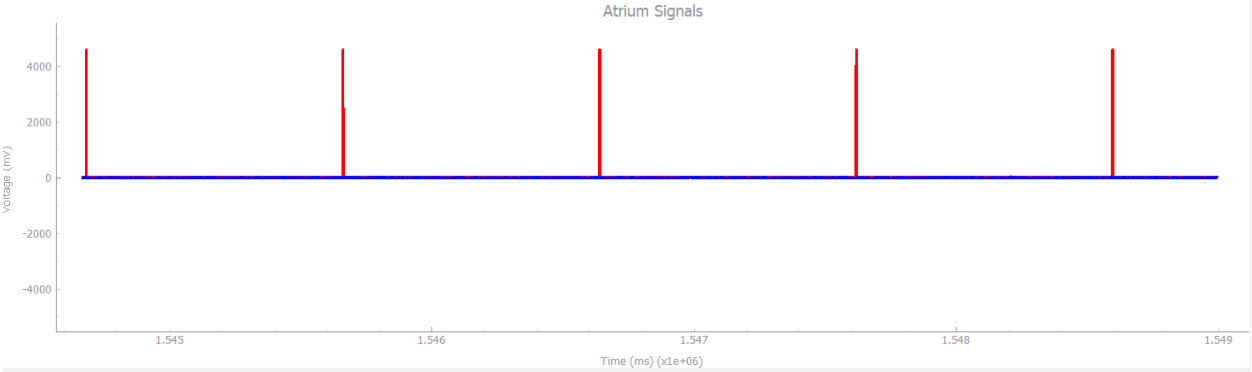

Figure 83: AAI Test Case 2 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = ON, BPM = 61

AAI - Case 3

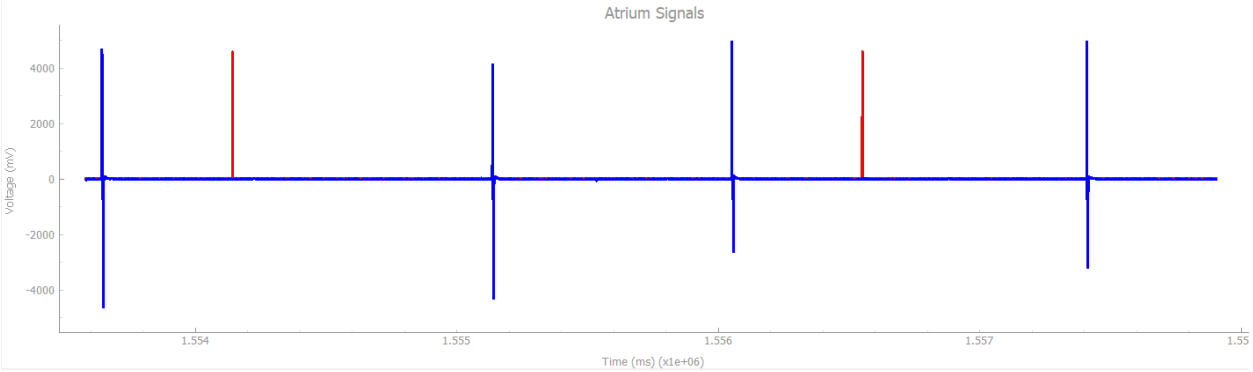

Figure 84: AAI Test Case 3 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = ON, BPM = 40

VVI - Case 1

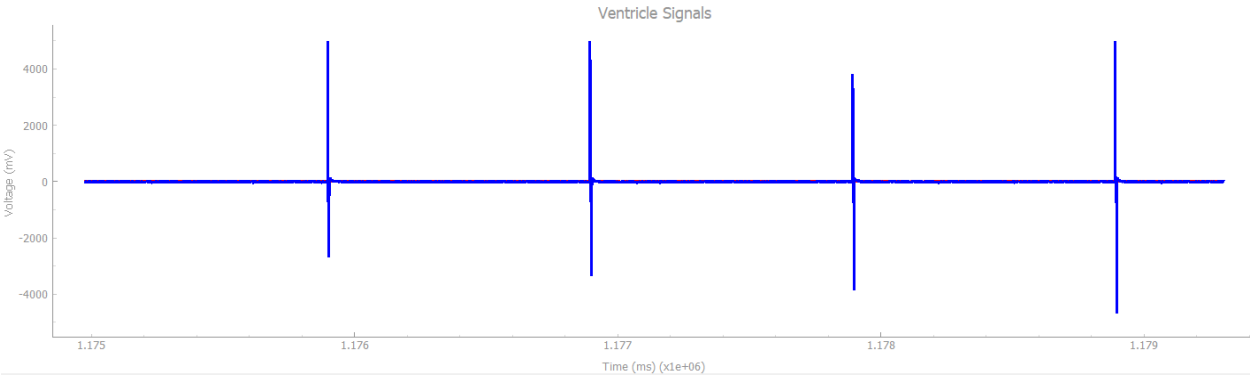

Figure 85: VVI Test Case 1 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = OFF

VVI - Case 2

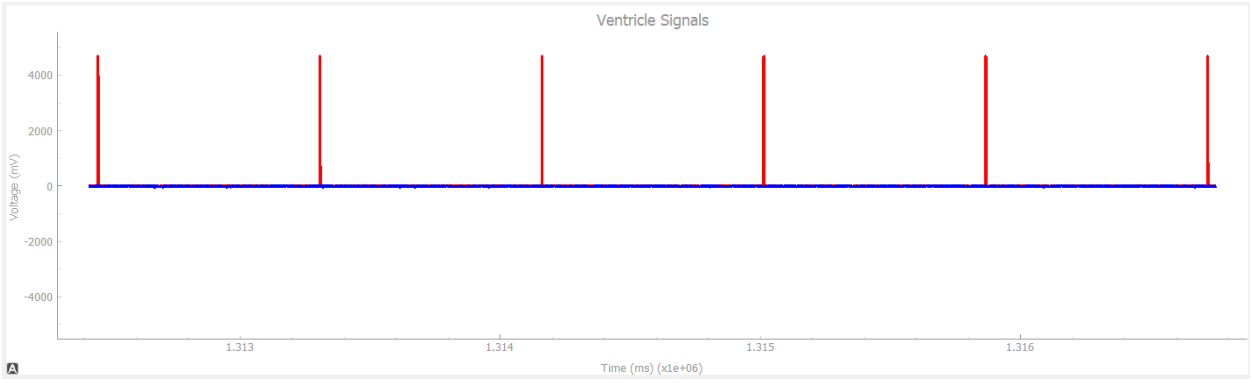

Figure 86: VVI Test Case 2 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 70

VVI - Case 3

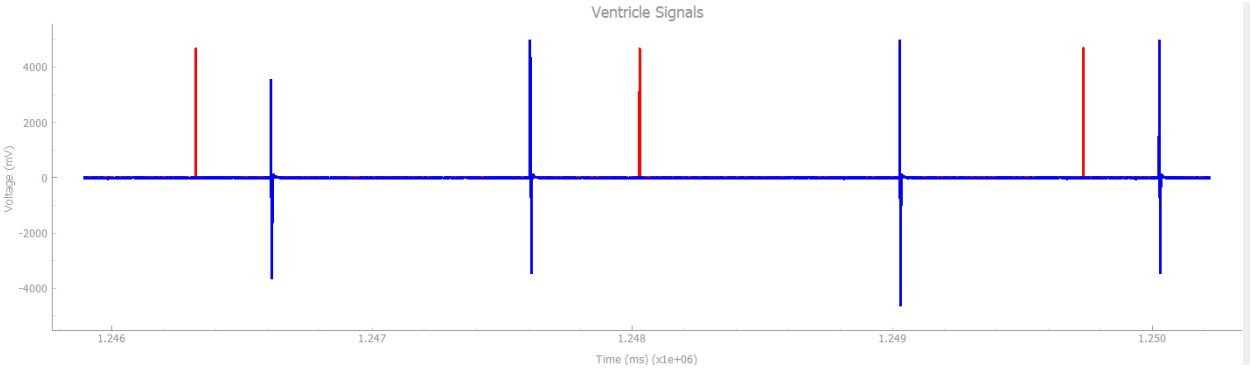

Figure 87: VVI Test Case 3 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 35

Additional Tests

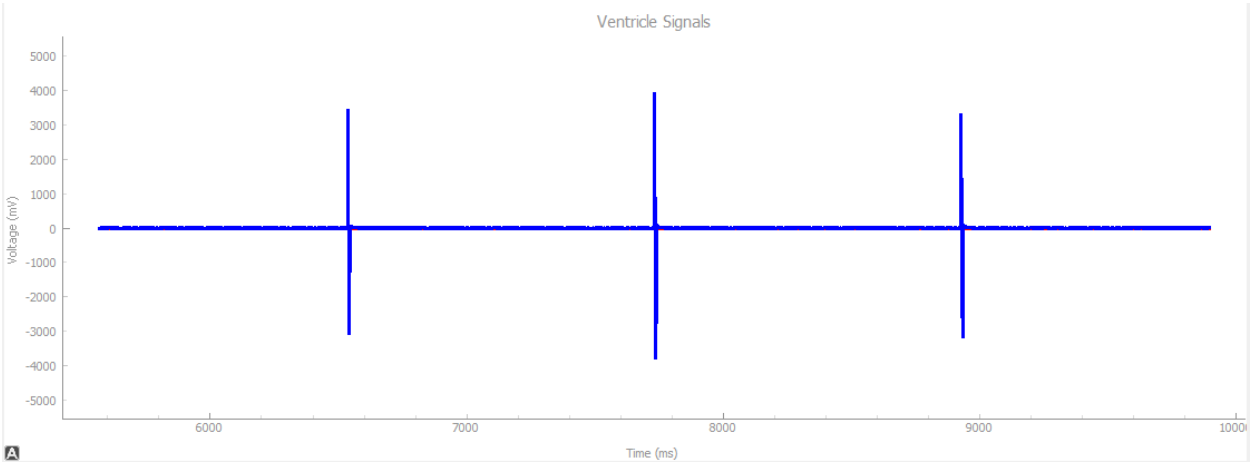

Figure 88: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = OFF

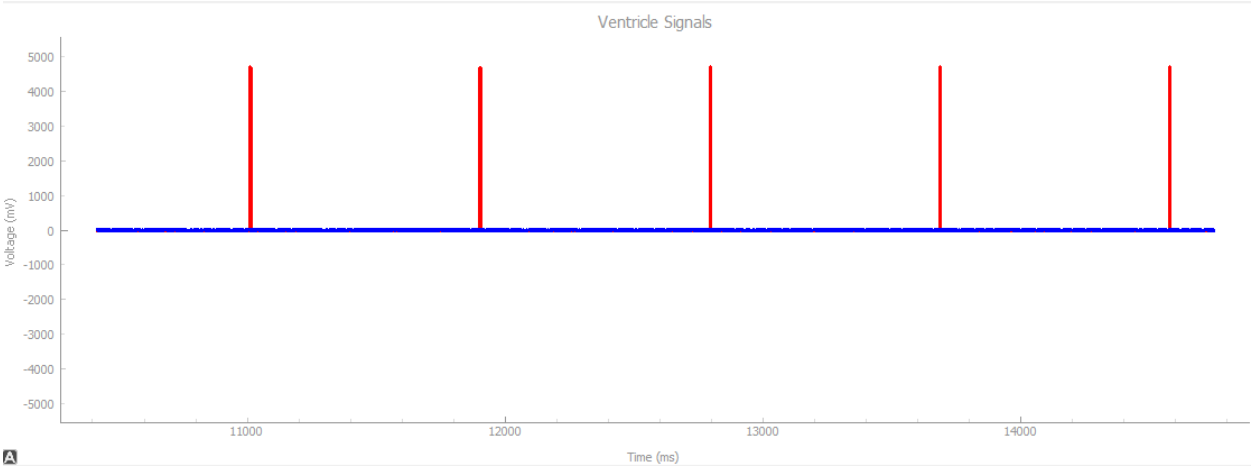

Figure 89: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 67 BPM

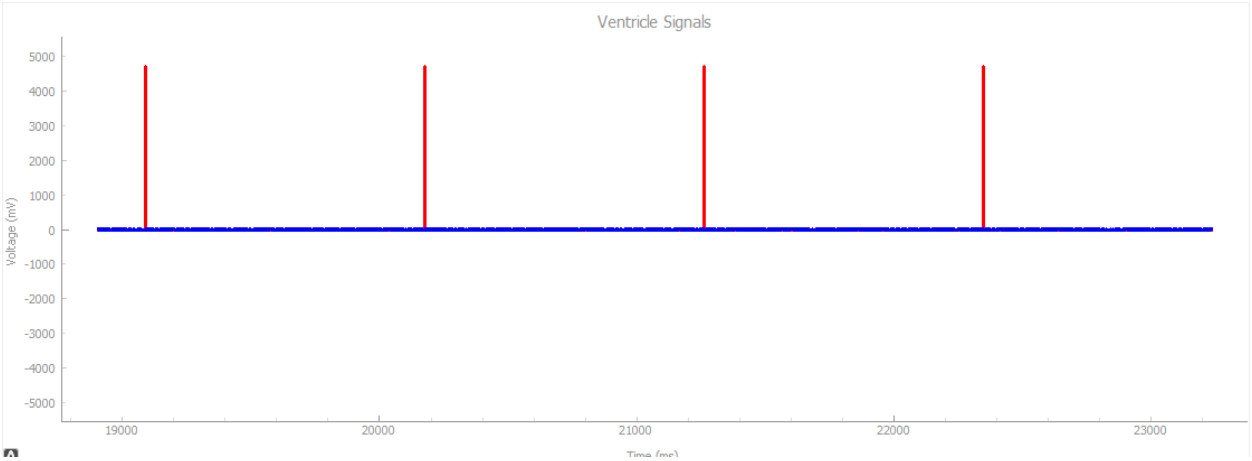

Figure 90: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 55 BPM

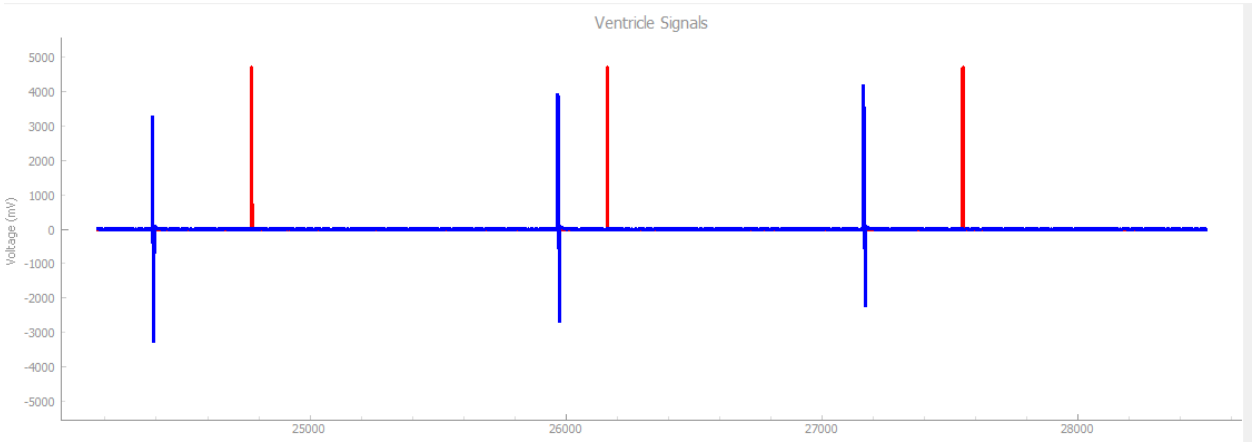

Figure 91: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 43 BPM

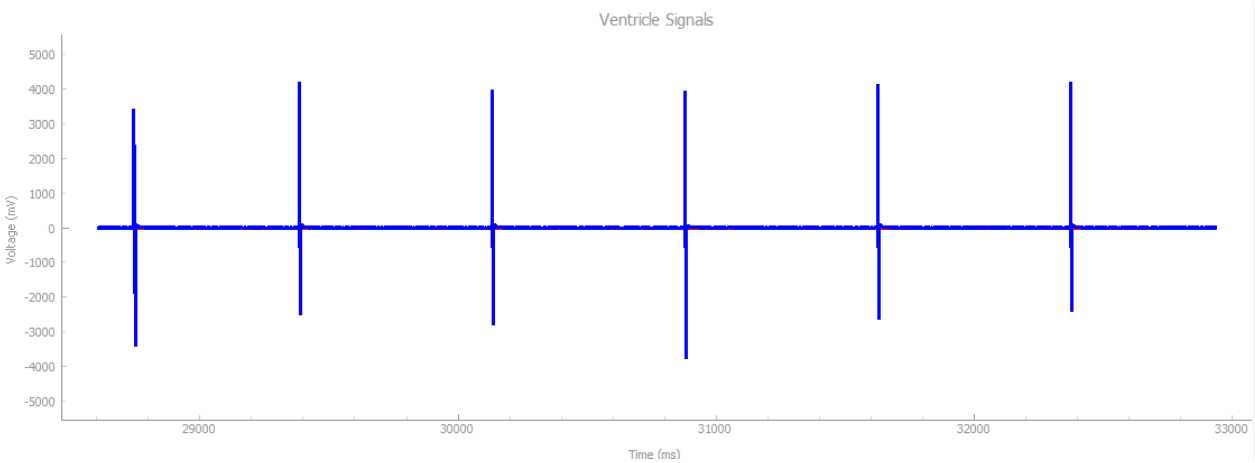

Figure 92: Mode - VVI, Ventricle Amp = 5, LRL = 80, Ventricle Pulse Width = 5, Natural Ventricle = OFF

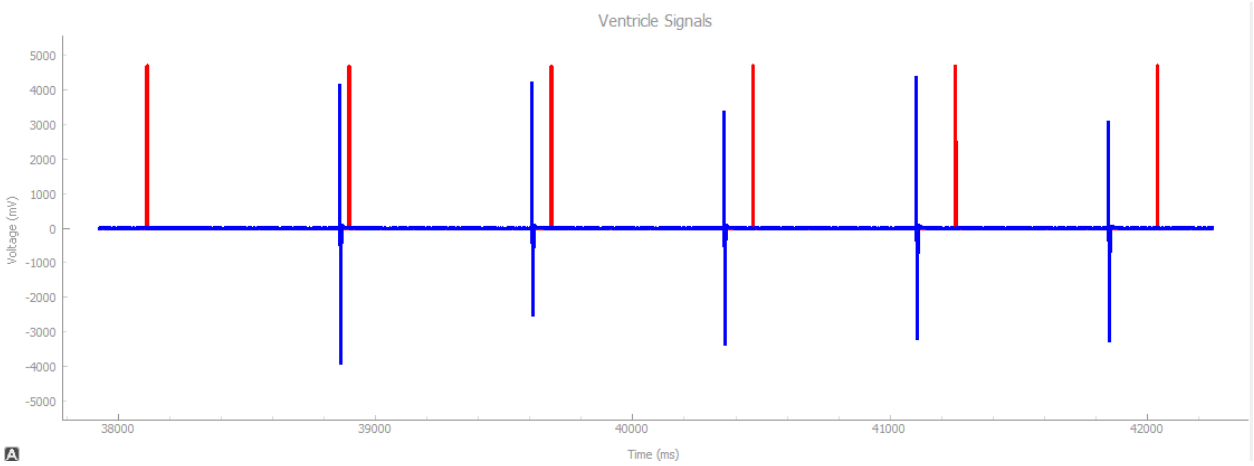

Figure 93: Mode - VVI, Ventricle Amp = 5, LRL = 80, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 76 BPM

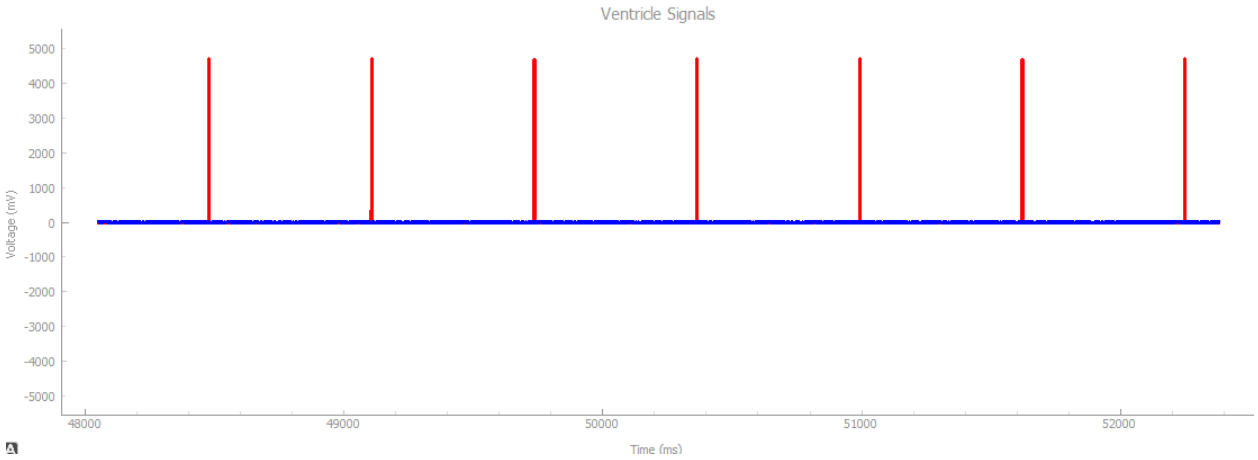

Figure 94: Mode - VVI, Ventricle Amp = 5, LRL = 80, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 95 BPM

Rate Adaptive Testing

Setting #1

MSR = 120, URL = 120, Mode = AOOR, Activity Threshold = Nominal, Recovery Time = Nominal, Response Factor = Nominal, Response Time = Nominal

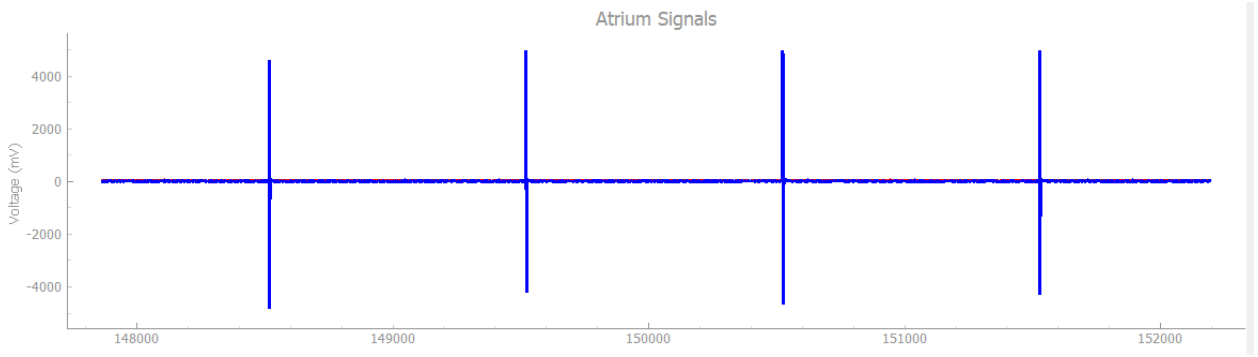

Figure 95: Before shaking

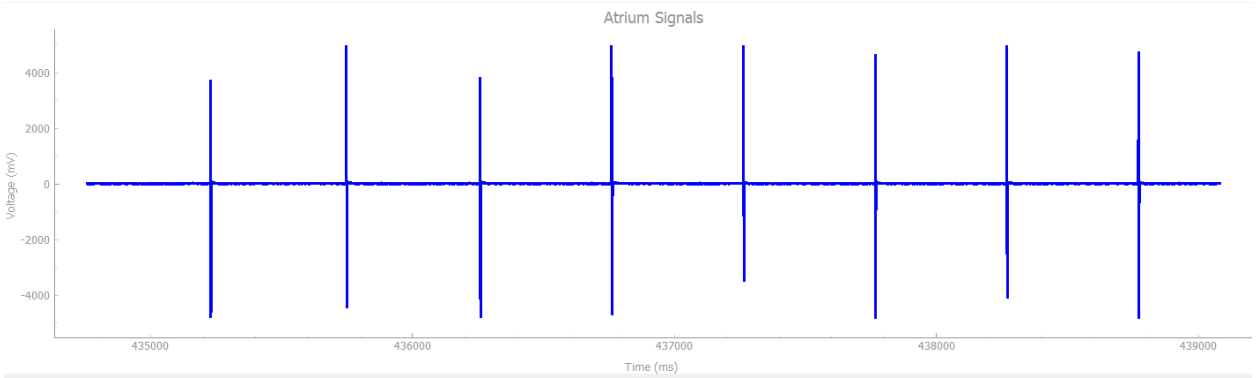

Figure 96: After shaking

Setting #2

MSR = 160, URL = 120, Mode = VOOR, Activity Threshold = Nominal, Recovery Time = Nominal, Response Factor = Nominal, Response Time = Nominal

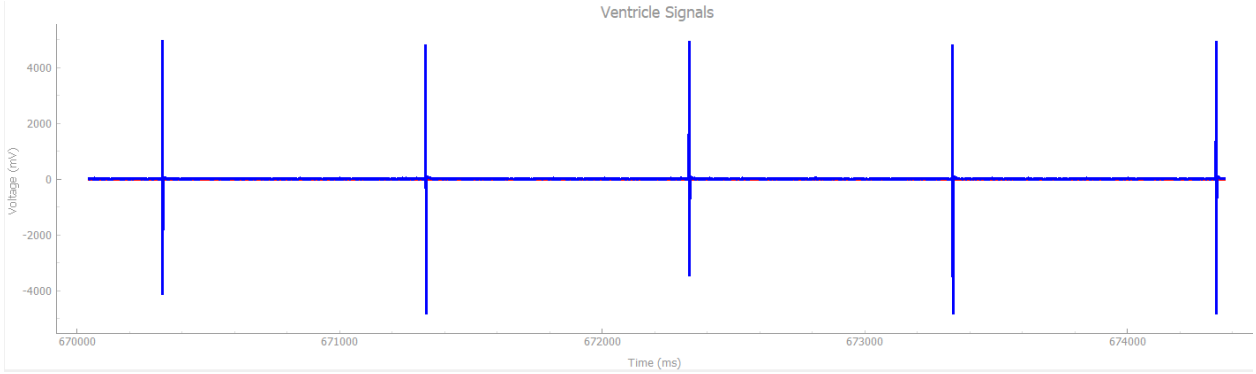

Figure 97: Before shaking

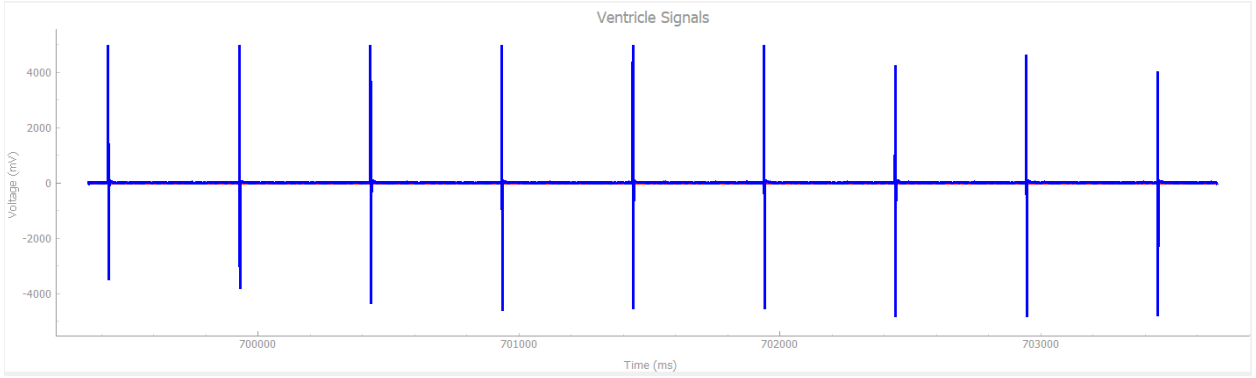

Figure 98: After shaking

Setting #3

Atrial Sensitivity = 5.3, Natural Atrial Heartbeat = 75 bpm, Activity Threshold = High, Response Factor = Nominal, Recovery Time = Nominal, Response Time = Nominal, Mode = AIIR

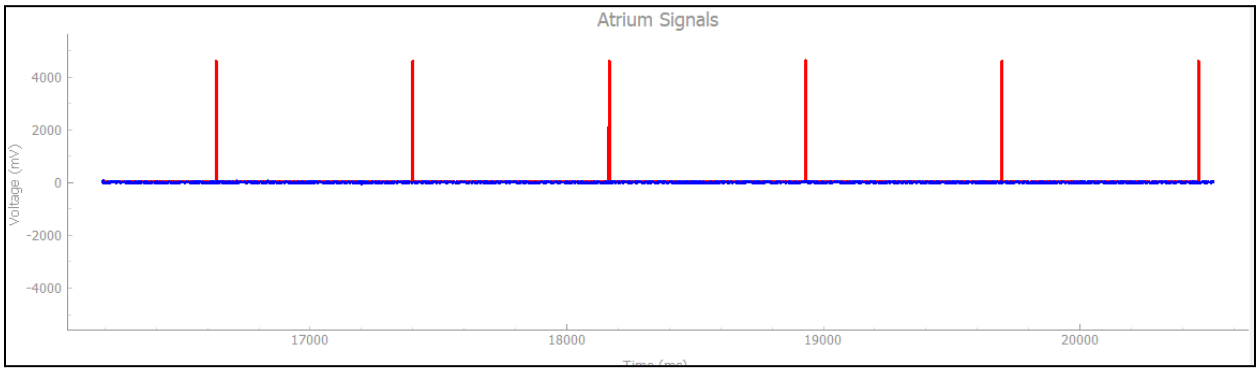

Figure 99: Before shaking

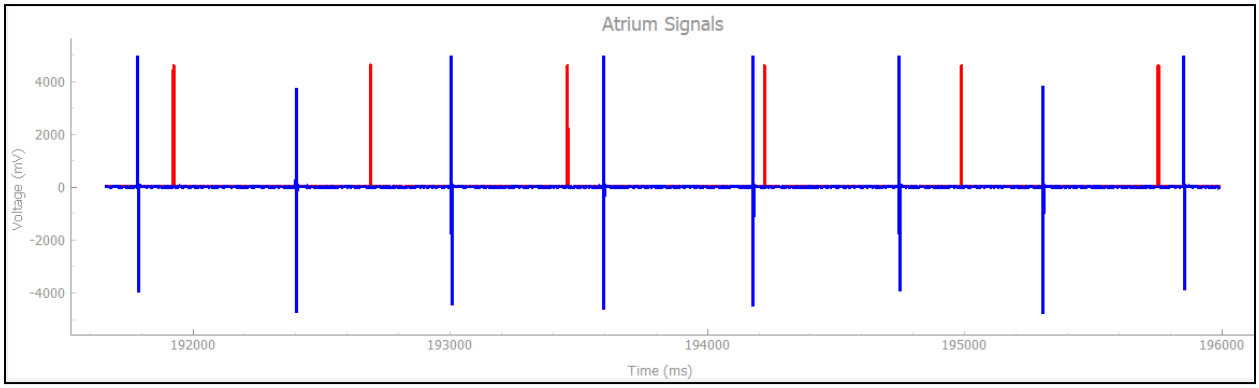

Figure 100: After shaking

Setting #4

Recovery time = 13, Natural Atrial Heartbeat = 100, Atrial Sensitivity = 5.3, Response Factor = Nominal, Activity Threshold = Nominal, Response Time = Nominal, Mode = AIIR

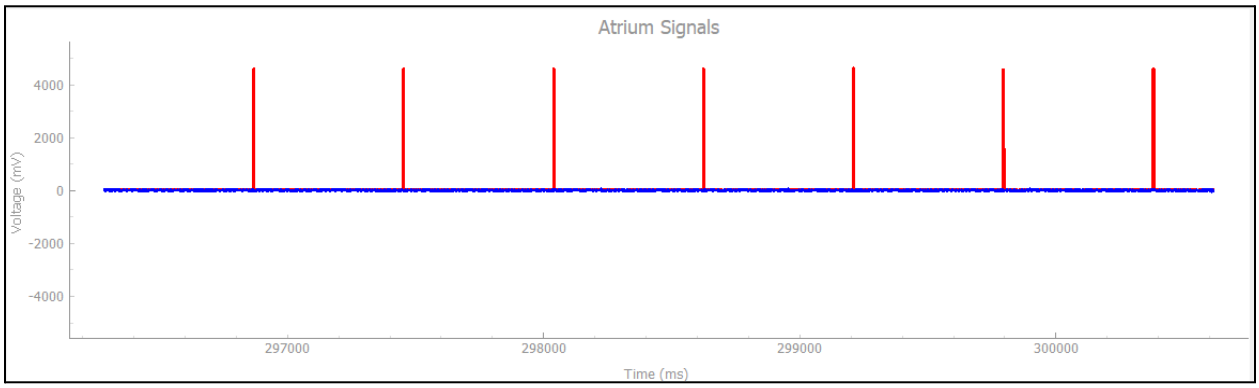

Figure 101: Before shaking

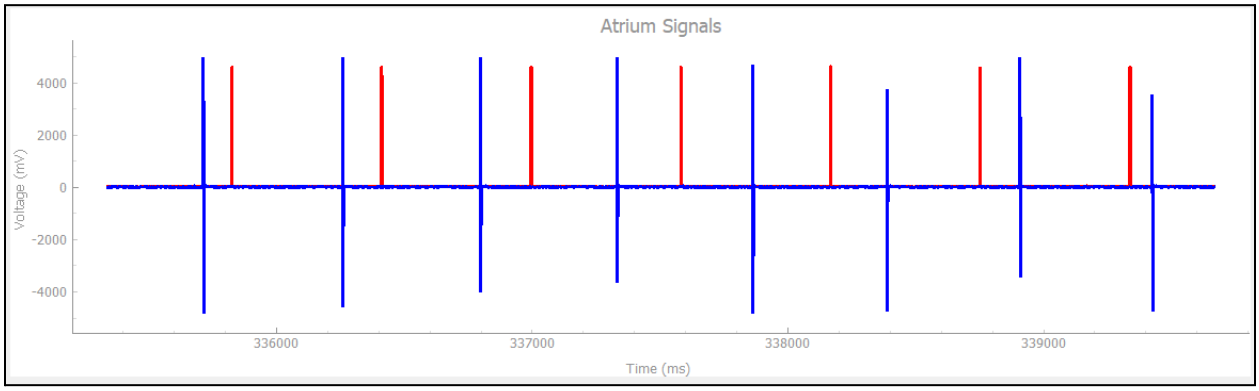

Figure 102: After shaking

Setting #5

Response time = 20s, Natural Ventricle Heartbeat = 40, Ventricle Sensitivity = 5.5, Activity Threshold = Nominal, Recovery Time = Nominal, Response Factor = Nominal, Mode = VIIR

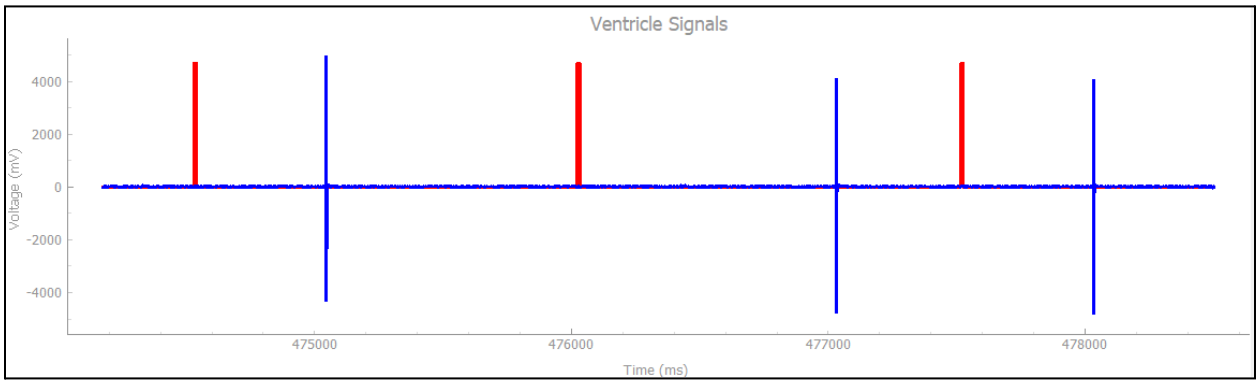

Figure 103: Before shaking

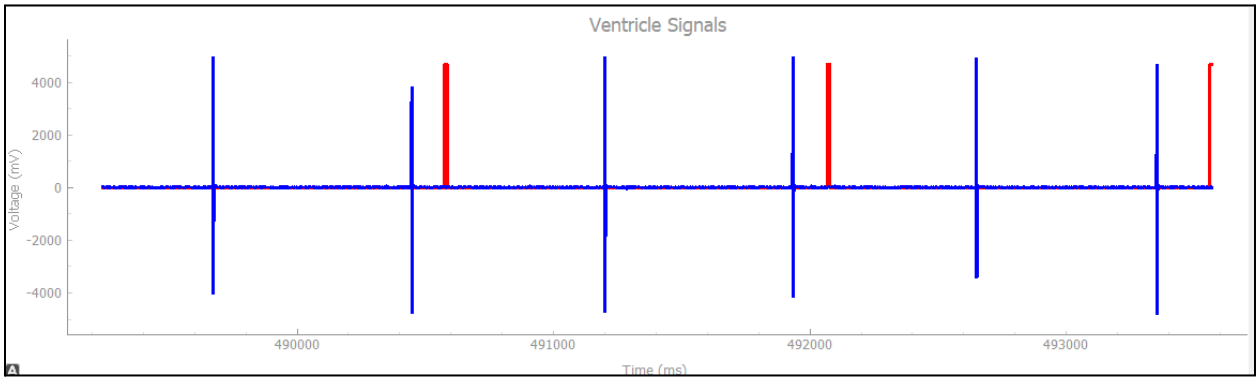

Figure 104: After shaking

Setting #6

Response Factor = 12, Natural Ventricle Heartbeat = 50, Ventricle sensitivity = 5.5, Activity Threshold = Nominal, Recovery Time = Nominal, Response Time = Nominal, Mode = VIIR

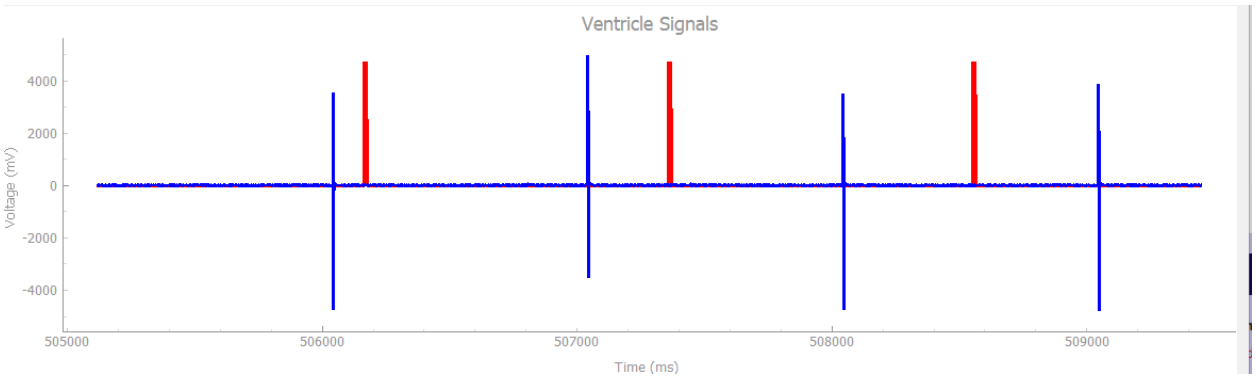

Figure 105: Before shaking

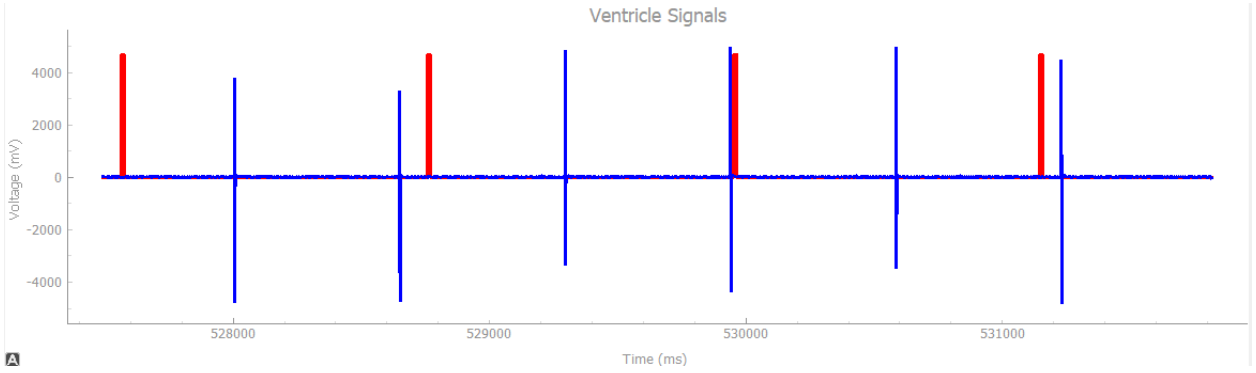

Figure 106: After shaking

Assurance Cases

#	Claims	Assumptions	Strategy & Argument	Evidence
1	The DCM and pacemaker work together to be a safe device for patients, and healthcare providers.	The DCM and pacemaker meet all project requirements		See all test cases
1.1	User registration and login works with no problems	A new user can create a new account, with a doctor key.	**Strategy:** Test account creation and login processes with	Registration - Case 1: Read all test cases from registration

		Used usernames are not allowed. Login only works with a existing username and matching password	valid and invalid credentials. **Argument:** If only valid credentials allow login and invalid inputs are rejected, the system is secure and functional.	case 1 - login case 4
1.2	The DCM transitions smoothly between interface pages (login, registration, main).	Each page is loaded dynamically based on the current user state. No unauthorized access to the main page.	**Strategy:** Trigger transitions between pages and observe for errors. **Argument:** If transitions occur without glitches or unauthorized access, the DCM manages interfaces correctly.	Login - Case 4: And Mainpage Test Cases
1.3	Pacing mode parameters are correctly initialized, displayed, and saved.	Default pacing mode values are accurate. User modifications to parameters are validated.	**Strategy:** Modify pacing parameters and save them. Reload the data to check for consistency. **Argument:** If saved parameters are correctly loaded and displayed, the system handles pacing data properly.	Checking Saved Parameters When Switching Modes: Checking Saved Parameters When signing out and signing back in: Checking Reset to Nominal Button:
1.4	Device connection and communication status are updated in real-time.	The pacemaker device ID is valid and compatible.	**Strategy:** Simulate device disconnection and reconnection	Checking Device Status:

		Serial port is available for data exchange.	while observing status indicators. **Argument:** If the DCM updates statuses in real-time, the communication management is functional.	Checking Send to Pacemaker Button/Communication Status:
1.5	Live graph widget accurately plots and resets pacing data, all buttons function as required.	Incoming data is within the expected format and range, communication with the pacemaker is faultless. Absolutely no errors with the data	**Strategy:** Showing the start, pause and save plot features. **Argument:** If graphs are accurate and reset works as expected, the plotting functionality is correct.	Checking Graph:
2	The pacemaker implements all the eight modes correctly.	The eight modes meet the requirements outlined in Pacemaker Requirements.	Strategy: Select UI mode tabs and observe pacemaker behaviour on HeartView. Argument: If the correct chamber is pacing and expected mode behaviour is observed on Heartview, then it is changing modes correctly.	Refer to testing results pacemaker
2.1	AOO is correctly implemented.	The programmable parameters needed for the mode, such as refractory period, are sent via serial	Strategy: Test the mode with natural heartbeat on and off.	AOO Test Case 1 AOO Test Case 2

		communication from the DCM. Accuracy of pacing is verified using Heartview.	Argument: If AOO paces the atrium at the correct rate regardless of the natural heart rate, it is working correctly.	
2.2	VOO is correctly implemented.	The programmable parameters needed for the mode, such as refractory period, are sent via serial communication from the DCM. Accuracy of pacing is verified using Heartview.	Strategy: Test the mode with natural heartbeat on and off. Argument: If VOO paces the ventricle at the correct rate regardless of the natural heart rate, it is working correctly.	VOO Test Case 1 VOO Test Case 2
2.3	AAI is correctly implemented.	The programmable parameters needed for the mode, such as refractory period, are sent via serial communication from the DCM. Accuracy of pacing is verified using Heartview.	Strategy: Adjust the natural heartbeat on Heartvide to see if the pacemaker is pacing. Argument: If AAI starts pacing when the natural heart rate is less than the lower rate limit, it is sensing and pacing correctly.	AAI Test Case 1 AAI Test Case 2 AAI Test Case 3
2.4	VVI is correctly implemented.	The programmable parameters needed for the mode, such as refractory period, are sent via serial communication from	Strategy: Adjust the natural heartbeat on Heartview to see if pacing occurs.	VVI Test Case 1 VVI Test Case 2 VVI Test Case 3

		the DCM. Accuracy of pacing is verified using Heartview.	Argument: If there is ventricular pacing during missed natural heartbeats, it is sensing and pacing correctly.	
2.5	All rate adaptive modes (AOOR, VOOR, AAIR, VVIR) are correctly implemented.	The programmable parameters needed for the mode, such as refractory period, are sent via serial communication from the DCM. Accuracy of pacing is verified using Heartview.	Strategy: Shake the board and determine on Heartview if pacing frequency increases within a few seconds. Argument: If pacing frequency increases after shaking the pacemaker, rate adaptive modes are correct.	Rate Adaptive Test Cases

Future Project Changes

DCM

Assignment 1

- In the future, several planned improvements have been discussed to enhance both the UI and functionality of the DCM.
- The software will support a couple of languages, to broaden accessibility for users in different regions and allowing healthcare practitioners to use the system locally in their preferred language.
- The UI will be redesigned to be more visually appealing and user-friendly, since the primary focus in this version was functionality. Future updates will focus on enhancing the look and feel to create a more intuitive user experience.
- Switching to an SQL-based database might be plausible, it would make it easier to index saved data and be less computationally taxing than the current setup. This change will improve data retrieval speed and reduce resource consumption as data volumes increase.

- The DCM will also be able to receive egram data and send it to the pacemaker, with real-time graphing capabilities to visualize these values. This will provide healthcare professionals with immediate insights into heart activity, improving monitoring accuracy.
- Additional features will include user deletion options, allowing administrators to manage accounts more effectively.
- Security enhancements:
    - Password hashing to protect user credentials and ensure data security.
    - Session timeout that automatically terminates the program if the computer is inactive for too long or loses connection with the pacemaker. This feature will prevent unauthorized access if the program is left open, adding an extra layer of safety for users.

These changes aim to make the DCM more secure, efficient, and user-friendly in future releases.

Assignment 2

- The software will support a couple of languages, to broaden accessibility for users in different regions and allowing healthcare practitioners to use the system locally in their preferred language.
- Additional features will include user deletion options, allowing administrators to manage accounts more effectively.
- Security enhancements are also a priority, including:
    - Password hashing to protect user credentials and ensure data security.
    - Session timeout that automatically terminates the program if the computer is inactive for too long or loses connection with the pacemaker. This feature will prevent unauthorized access if the program is left open, adding an extra layer of safety for users.

There are less future additions since previous ones were added to the current functionality while others were phased out. We prioritized serial communication, and UI improvements over everything else. Considering we are using 10 accounts, an sql database actually would have been slower than using the text file, causing unnecessary inefficiencies.

Simulink

Assignment 1

Although the first four pacing modes (AOO, VOO, AAI, VVI), as well as the DCM interface have been successfully implemented, there are still significant changes and additional features that must be implemented to achieve a fully functional pacemaker software, such as:

- **Implement rate-adaptive pacing modes (AOOR, VOOR, AAIR, VVIR)**
    - Rate adaptive pacing adjusts the pacing rate based on patient activity level, as measured using the pacemaker's onboard accelerometer.

- **Add additional parameters required by new modes**
    - For the new modes to function properly, more parameters must be included.
    - Examples - Maximum sensor rate (MSR), postventricular atrial refractory period (PVARP), activity threshold, reaction time, response factor, recovery time.
- **Modify the Simulink model to allow for communication with the DCM**
    - The pacemaker must be able to send parameter values from the DCM, as well as send egram reports generated to DCM.

Assignment 2

- **Addition of rate smoothing and hysteresis.** While these variables are already considered in the DCM, they can also be implemented in the Simulink model.
    - Hysteresis will limit unnecessary pacing, allowing the heart to beat mainly on its own, and only enable pacing when required by the heart. I.e, it enables self-pacing.
    - Rate smoothing will allow for a more controlled incrementation when increasing and decreasing the heart rate in the Rate Adaptive subsystem.

References:

[1] E. Laskowski, “2 easy, accurate ways to measure your heart rate,” Mayo Clinic, https://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979 (accessed Oct. 23, 2024).
[2] S. Choi, M. Baudot, O. Vivas, and C. M. Moreno, “Slowing down as we age: Aging of the Cardiac Pacemaker’s Neural Control,” GeroScience, vol. 44, no. 1, pp. 1–17, Jul. 2021. doi:10.1007/s11357-021-00420-3
[3] “Understanding the basics of pulse width modulation (PWM) - technical articles,” Control, https://control.com/technical-articles/understanding-the-basicsof-pulse-width-modulation-pwm/ (accessed Oct. 25, 2024).
[4] “Bradycardia,” Mayo Clinic, https://www.mayoclinic.org/diseases-conditions/bradycardia/symptoms-causes/syc-20355474 (accessed Oct. 25, 2024).
[5] S. Khristich, “Medical Device Software Development: The complete guide,” TATEEDA, https://tateeda.com/blog/how-to-build-custom-medical-device-software-the-complete-guide (accessed Oct. 25, 2024).
[6] Sruthy, “Python vs C++ (top 16 differences between C++ and python),” Software Testing Help - FREE IT Courses and Business Software/Service Reviews, https://www.softwaretestinghelp.com/python-vs-cpp/ (accessed Oct. 25, 2024).