

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.....	3
Introduction.....	4
Research & Planning.....	5
C++.....	6
Python.....	6
PyQt.....	7
Requirements:.....	7
DCM Requirements.....	7
Pacemaker Requirements.....	8
Simulink Design:.....	9
Summary.....	9
Variables & Constants.....	13
Design Considerations.....	15
Input Subsystem.....	16
Mode Decision.....	18
AOO.....	20
VOO.....	20
AAI.....	21
VVI.....	22
Output Subsystem.....	22
DCM Design:.....	23
Design Considerations.....	23
Class Diagram.....	29
Classes.....	29
Methods.....	30
Testing Results - DCM.....	35
Registration - Case 1:.....	35
Registration - Case 2:.....	36
Registration - Case 3:.....	36
Registration - Case 4:.....	37
Registration - Case 5:.....	38
Login - Case 1:.....	38
Login - Case 2:.....	39
Login - Case 3:.....	39

Problems and Challenges:	48
DCM	48
Simulink	49
Testing Results - Pacemaker	50
AOO - Case 1	50
AOO - Case 2	50
VOO - Case 1	51
VOO - Case 2	51
AAI - Case 1	52
AAI - Case 2	52
AAI - Case 3	53
VVI - Case 1	53
VVI - Case 2	54
VVI - Case 3	54
Additional Tests	55
Future Project Changes	58
DCM	58
Simulink	58
References:	60

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

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**).

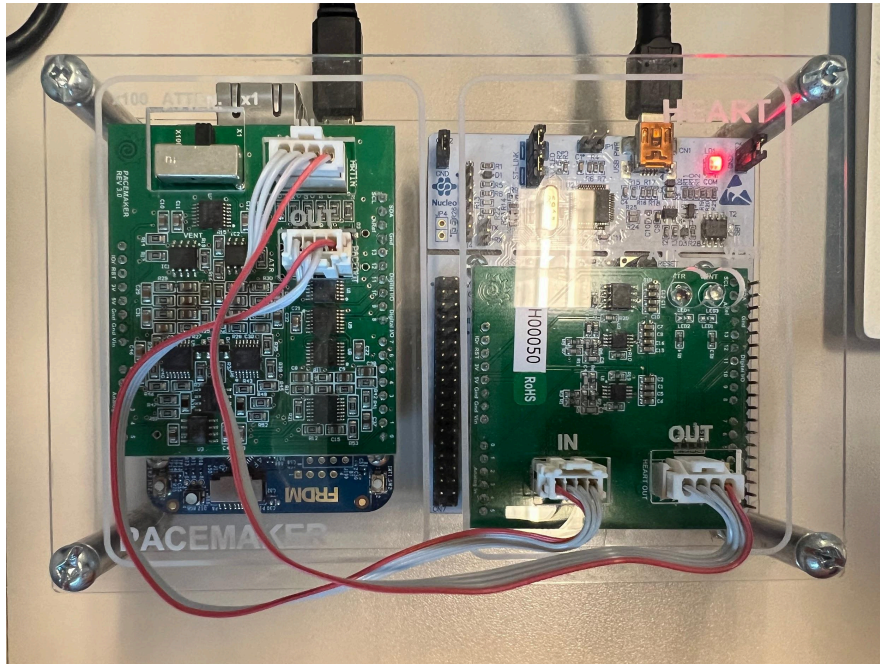


Figure 1. Image displaying the hardware kit showcasing the pacemaker (left) and the heart (right)

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 given 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 layout 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, and AAI. 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.

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

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 1: Required parameters grouped by pacing mode

Parameter	AOO	AAI	VOO	VVI
Lower Rate Limit	X	X	X	X
Upper Rate Limit	X	X	X	X
Atrial Amplitude	X	X		
Ventricular Amplitude			X	X
Atrial Pulse Width	X	X		
Ventricular Pulse Width			X	X
Atrial Sensitivity		X		
Ventricular Sensitivity				X
ARP		X		
VRP				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.

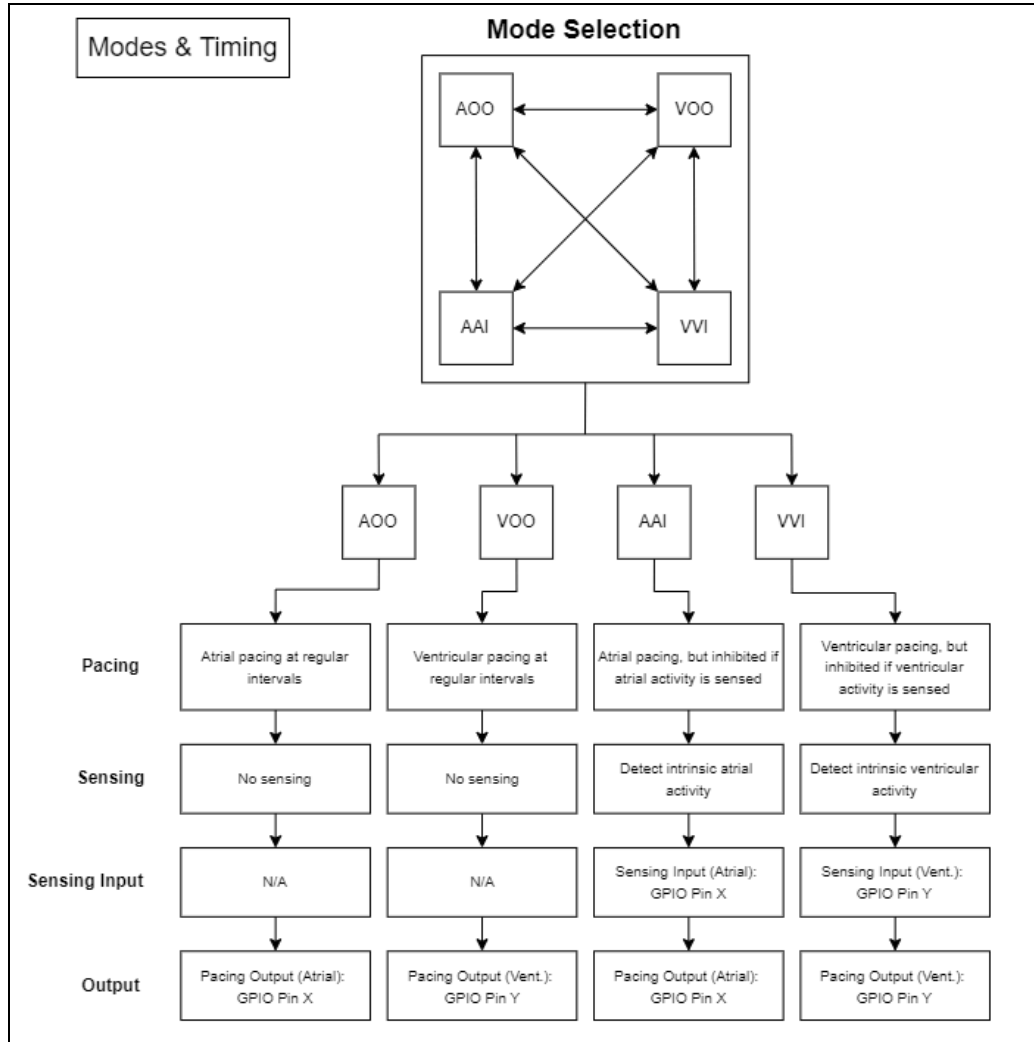


Figure 2: Mode selection logic for each of the four modes - AOO, VOO, AAI, and VVI,

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.

General Flow

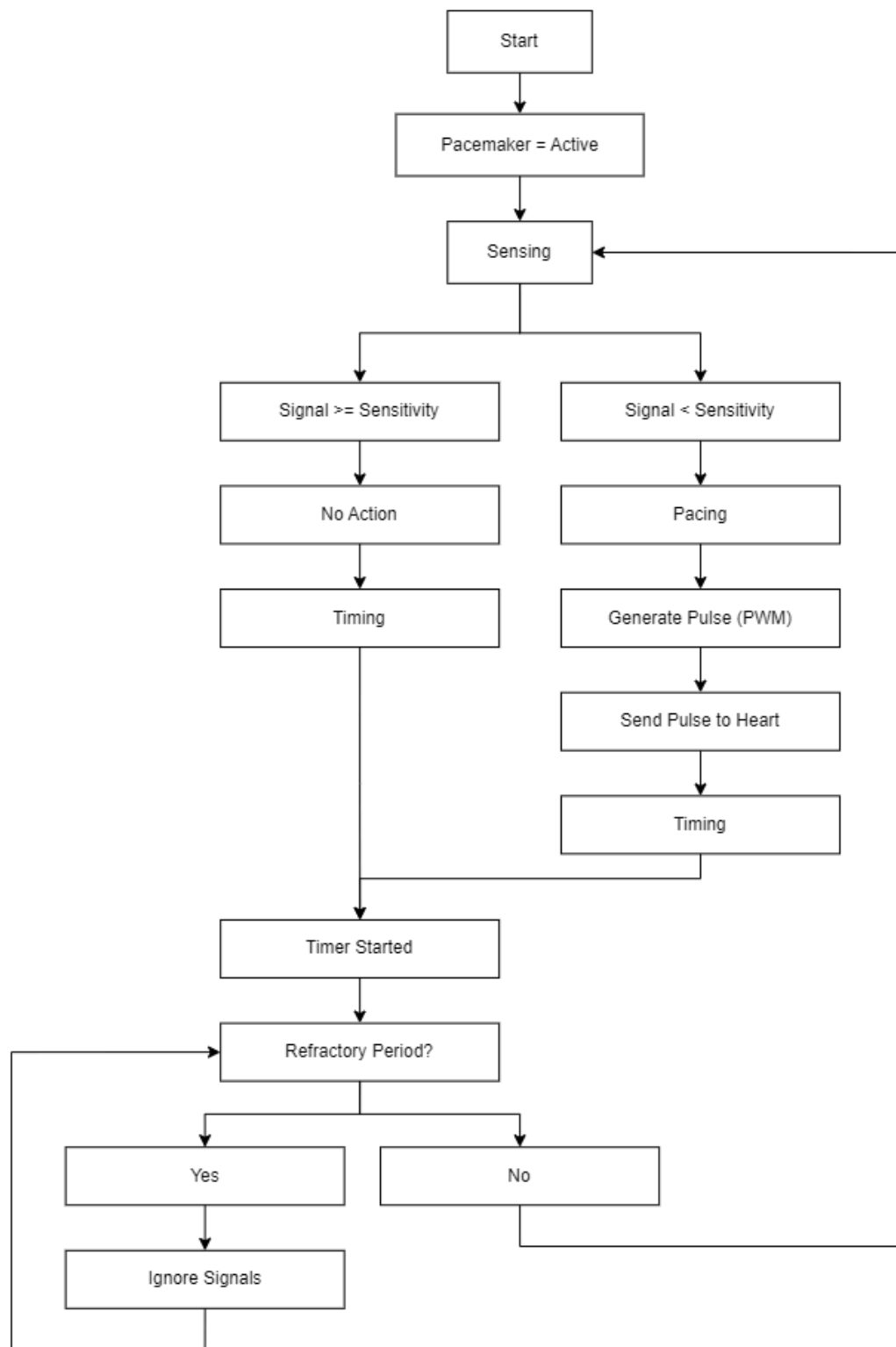


Figure 3: Flow of logic for sensing modes: AAI and VVI.

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 2: 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 circuitry. 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 as the indicator of Safety When 1, turns on indicating that one of the parameters is unsafe When 0, turns off

Variables & Constants

The variable definitions and values used in the input subsystem are described in **Table 3**.

Table 3: 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.	<ul style="list-style-type: none"> 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.	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • Current model value: 5 ms • Allowable range: 1-11 ms (HeartView)
Atrium_Amplitude	Atrium amplitude is the amplitude of the paced signal in the atria.	<ul style="list-style-type: none"> • Current model value: 5 Volts • Allowable range: 0.5-5 Volts
Atrium_RP	Atrium refractory period, which is the time in between each discharged pulse.	<ul style="list-style-type: none"> • Current 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 pacing interval.	<p>Is user adjusted, where the value will range based on different populations (based on factors including age) [2]</p> <ul style="list-style-type: none"> • Current 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.	<p>Is user adjusted, where the value will range based on different populations (based on factors including age) [2]</p> <ul style="list-style-type: none"> • Current model value: 120 ppm • Allowable range: 50-175 ppm
Ventricle_PW	Ventricle pulse width, which is the pulse width sent to the heart.	<ul style="list-style-type: none"> • Current model value: 5 ms • Allowable range: 1-11 ms (HeartView)
Ventricle_Amplitude	Ventricle amplitude is the amplitude of the paced signal in the ventricles.	<ul style="list-style-type: none"> • Current model value: 5 Volts • Allowable range: 0.5-5 Volts
Ventricle_RP	Ventricle refractory period, which is the time in between each discharged pulse	<ul style="list-style-type: none"> • Current 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	A constant block that should be set between 1-4 to activate one of the four modes.	<ul style="list-style-type: none"> • 1 - AOO • 2 - VOO • 3 - VVI • 4 - AAI

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.

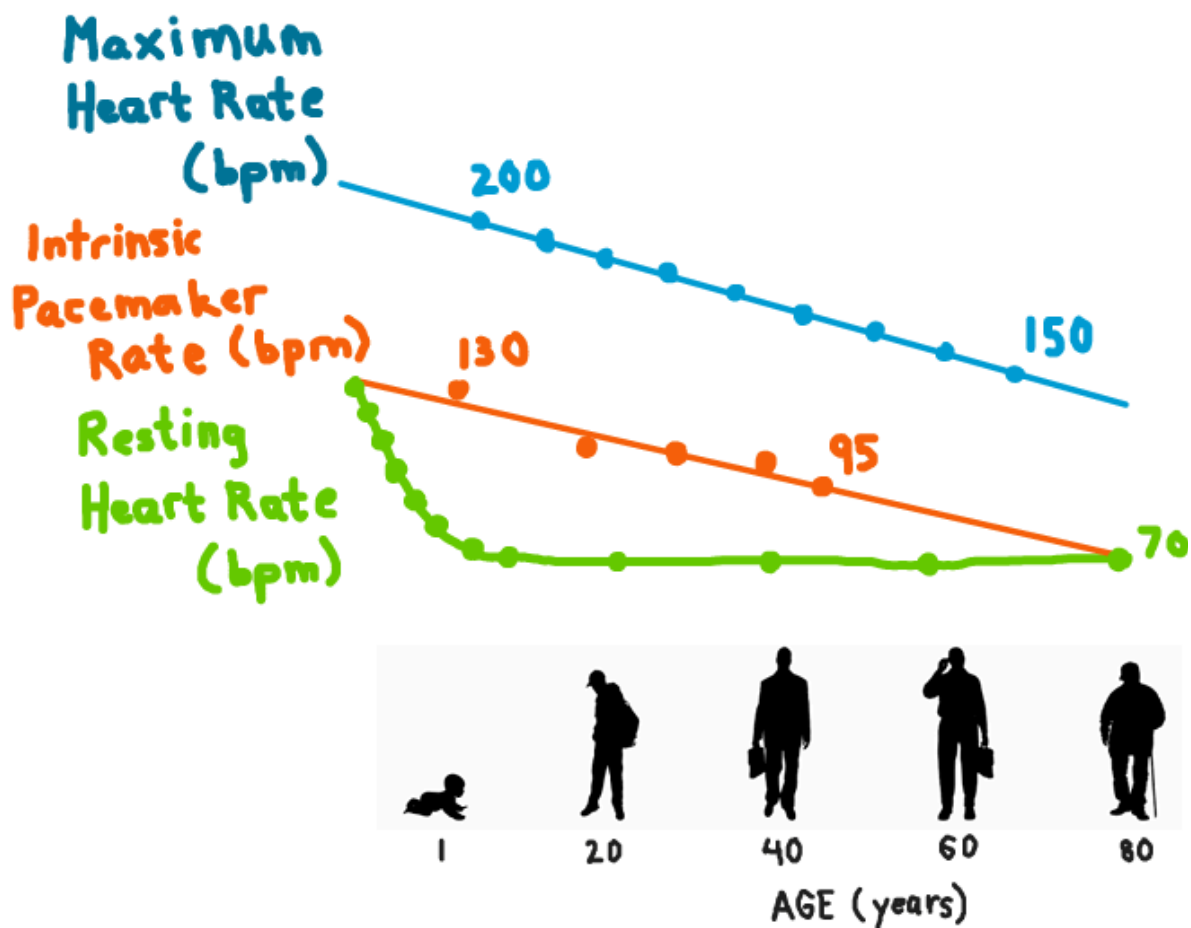


Figure 4: Average maximum and resting heart rate as age increases, adapted from [2].

Design Considerations

The design considerations for this pacemaker simulation focused on functionality, safety, and user control. Each of the four 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 four modes by selecting a corresponding value (1-4). A safety mechanism was implemented to prevent the system from running if an invalid mode is entered (outside of 1-4), ensuring the pacemaker remains inactive in these cases to prevent improper functioning.

Additionally, safety checks were integrated to monitor critical input parameters, as shown in **Figure 5**. If any of the following conditions are not met, the system will trigger a red LED indicator: lower rate limit less than 175, atrium amplitude less than seven, ventricle amplitude less than seven, atrium refractory period less than or equal to 500, or ventricle refractory period less than or equal to 500. The visual warning signals that the inputs are unsafe, and the pacemaker will not engage in any of the modes until these values are corrected. These design considerations ensure that the system operates safely and effectively, preventing the pacemaker from functioning under hazardous conditions.

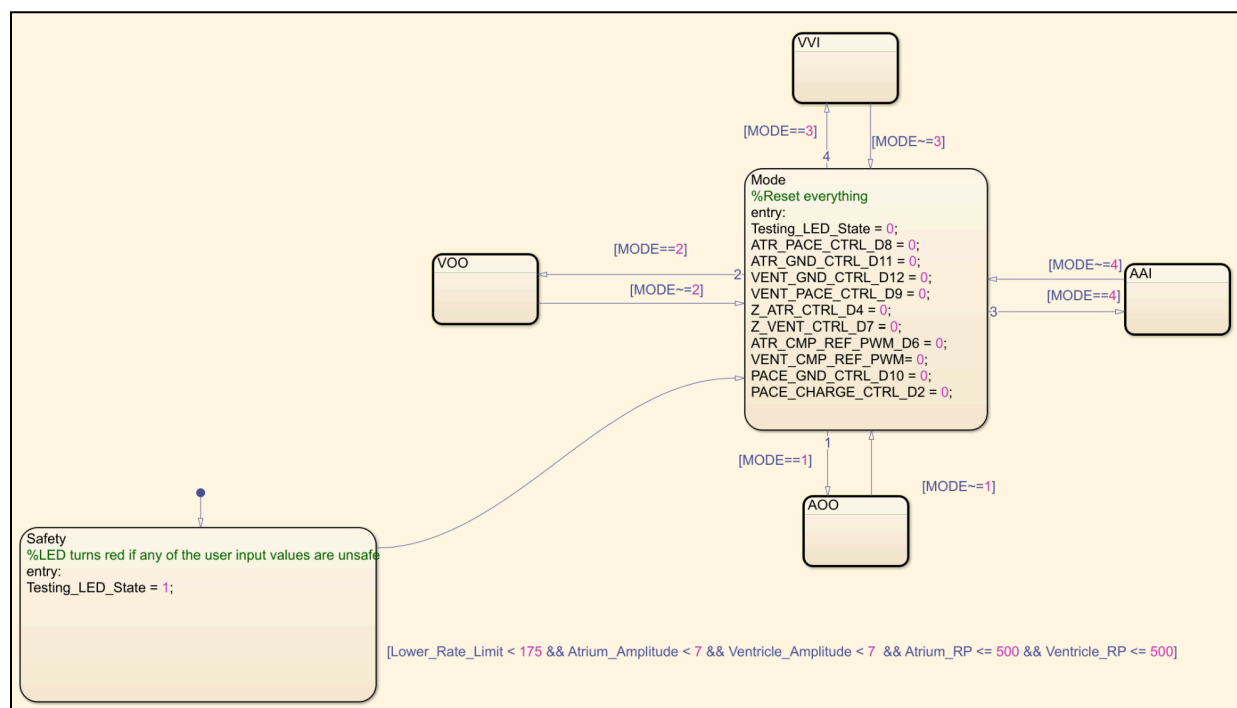


Figure 5: Safety block in bottom left corner is connected to the mode selection. As shown in the process arrow, if the parameters are beyond the bounds, a red LED will flash and no mode will be activated.

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 6**. 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. Mode-switching logic will later be implemented. The current model uses a constant block to activate different modes by inputting a value between 1-4. This will be further expanded in the [Mode Decision](#) section.

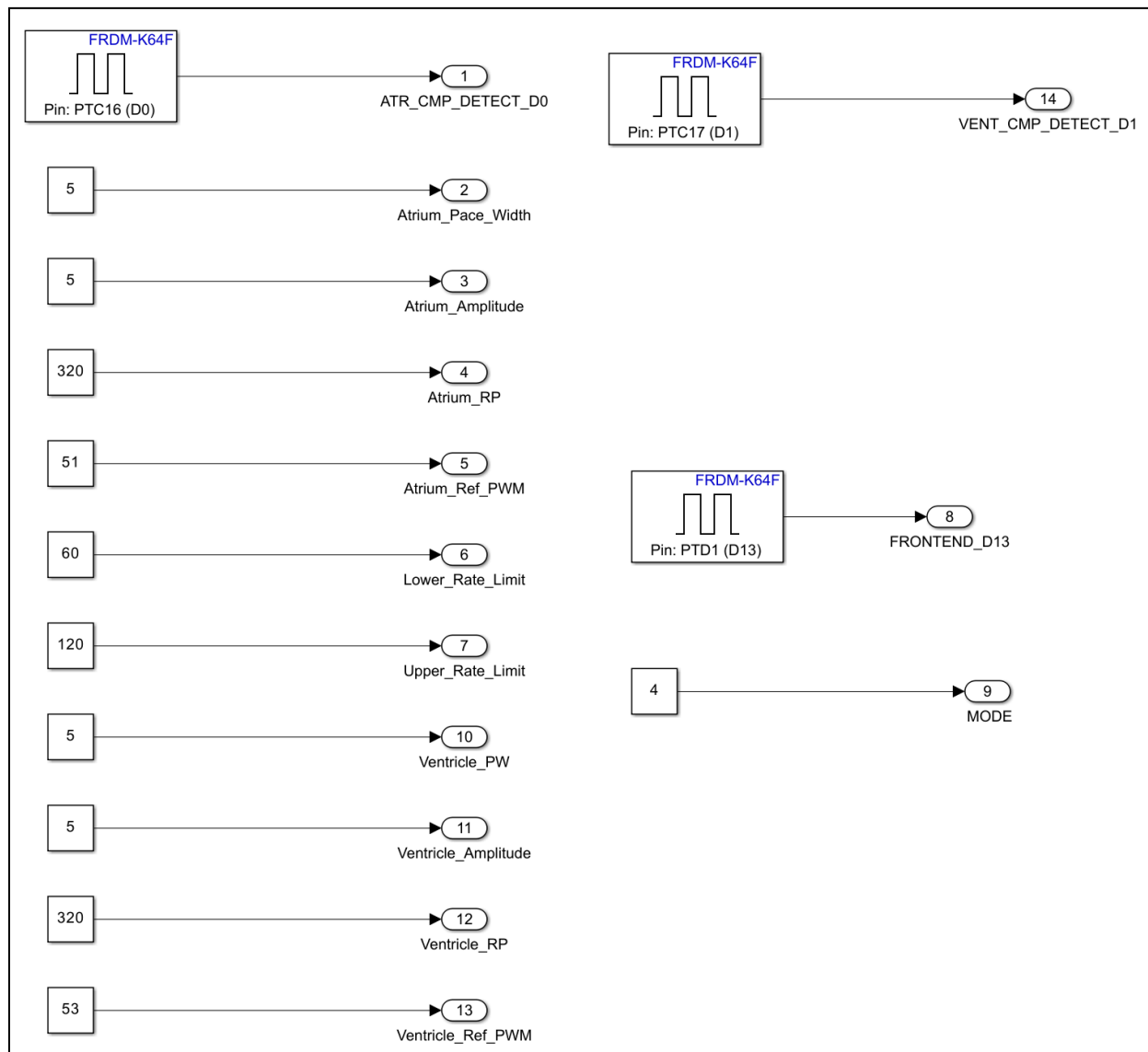


Figure 6: Input parameter subsystem that will be linked to the DCM for user adjustments.

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:

$$\text{PWM duty cycle} = 53/100 = 53\%$$

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

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 7**. This allows the detection of very slow signals under 60 bpm, a characteristic and symptom of bradycardia arrhythmia (a conductive disorder) [4].

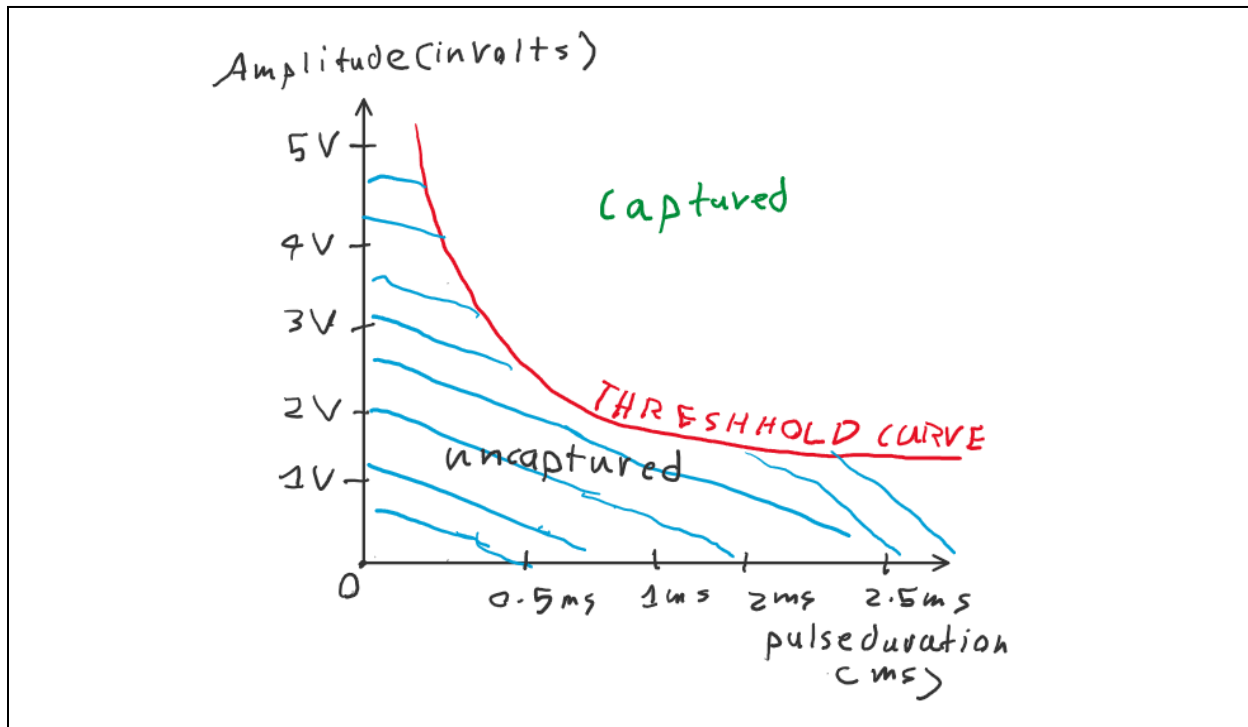


Figure 7: Estimation of the strength-duration curve showing the relationship between voltage and pulse width capture

However, when testing multiple cases of low natural heartbeats below 50 bpm on the HeartView software, the AAI and VVI 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 and VVI 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.

Mode Decision

After the input subsystem, the model moves into the Modes state flow diagram, which determines what mode the pacemaker will operate in based on the parameter specified in the input subsystem. However, before a mode can be chosen, the input parameters must be tested to ensure they are safe and will not harm the patient as shown in **Figure 8**. The lower rate limit, atrial and ventricular amplitude, and atrial and ventricular refractory period are tested to ensure they are within acceptable bounds.

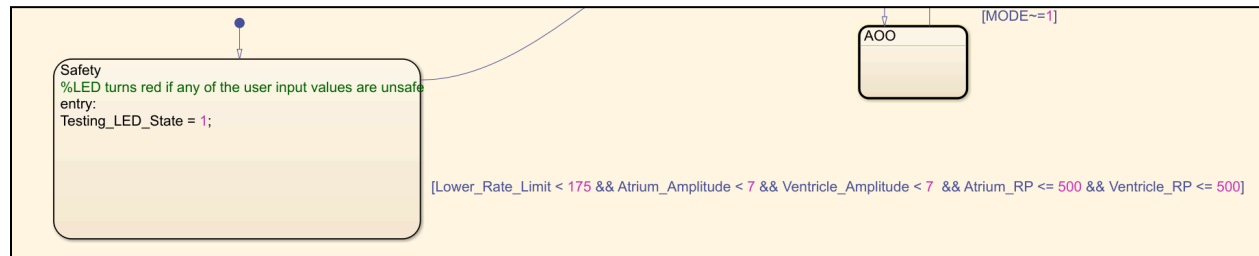


Figure 8: Logic to determine whether the input parameters are safe for use

If the parameters are determined to be safe, the simulation enters the Mode block, which first sets all the variables used by the modes to 0, as seen in **Figure 9**. 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 [VVI](#), and 4 is [AAI](#). 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 rests, and then enters the next mode diagram based on the value of the updated mode variable.

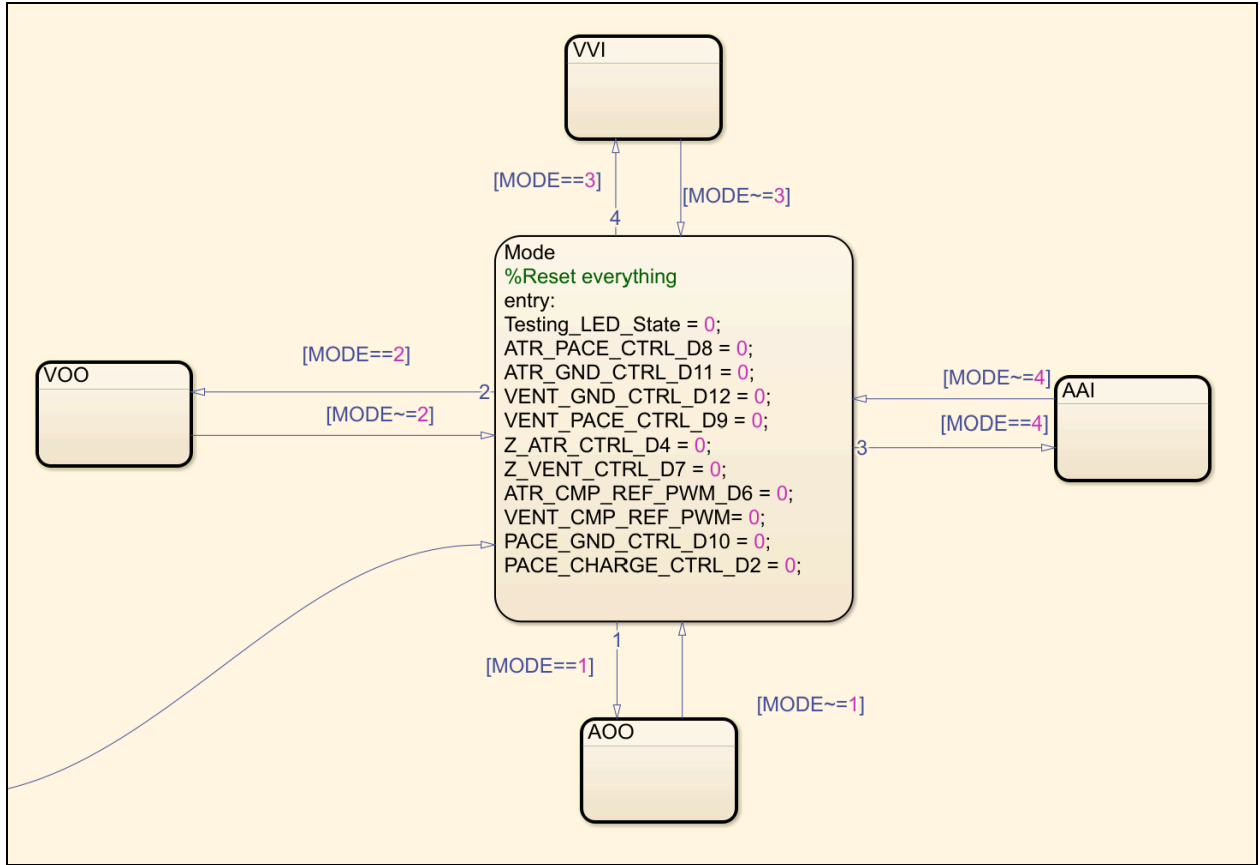


Figure 9: Logic for choosing which mode to activate (based on the MODE constant block from the input subsystem)

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 10**. 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.

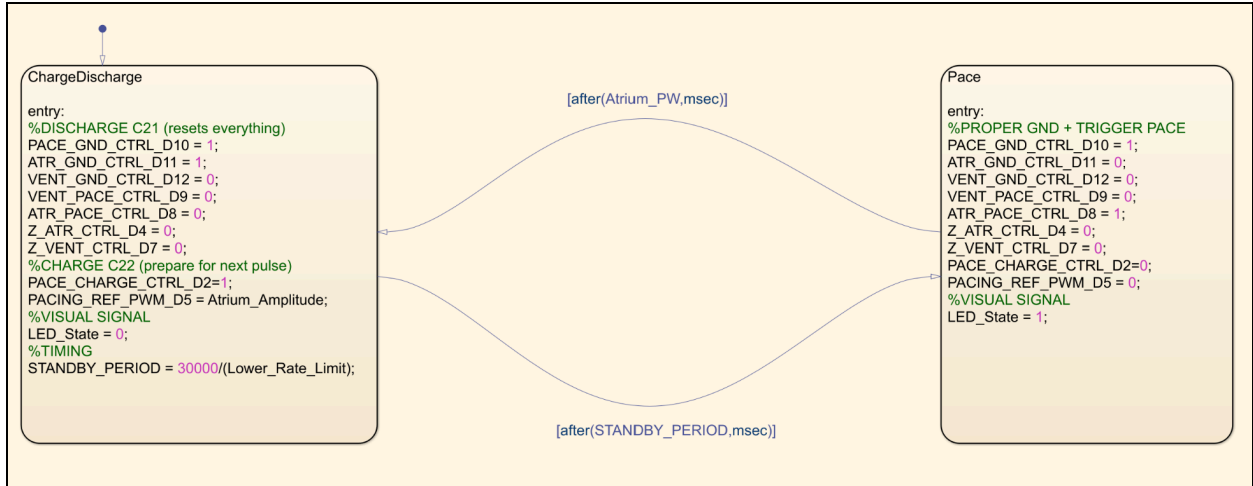


Figure 10: Stateflow diagram of AOO mode on Simulink

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.

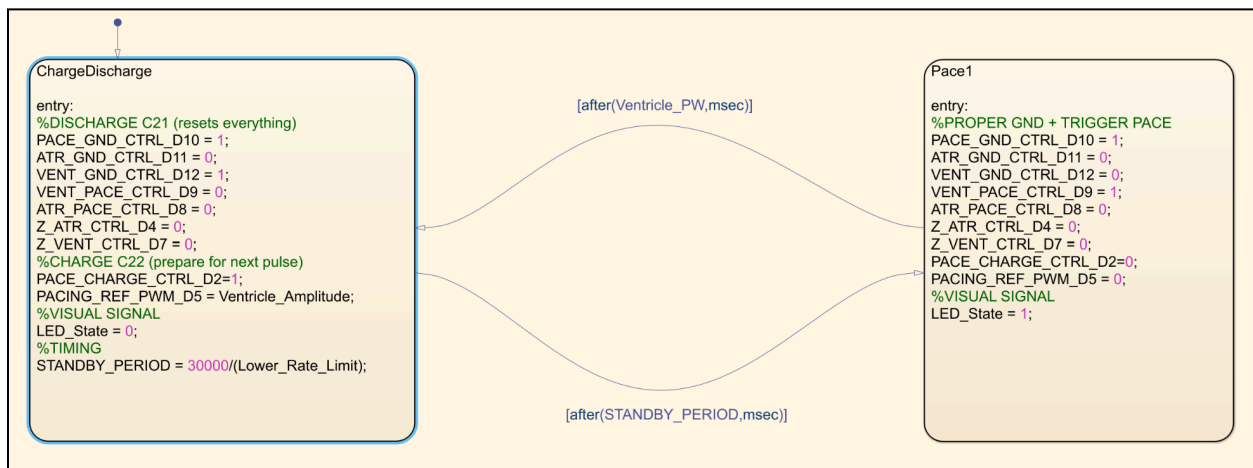


Figure 11: Stateflow diagram of VOO mode on Simulink

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.

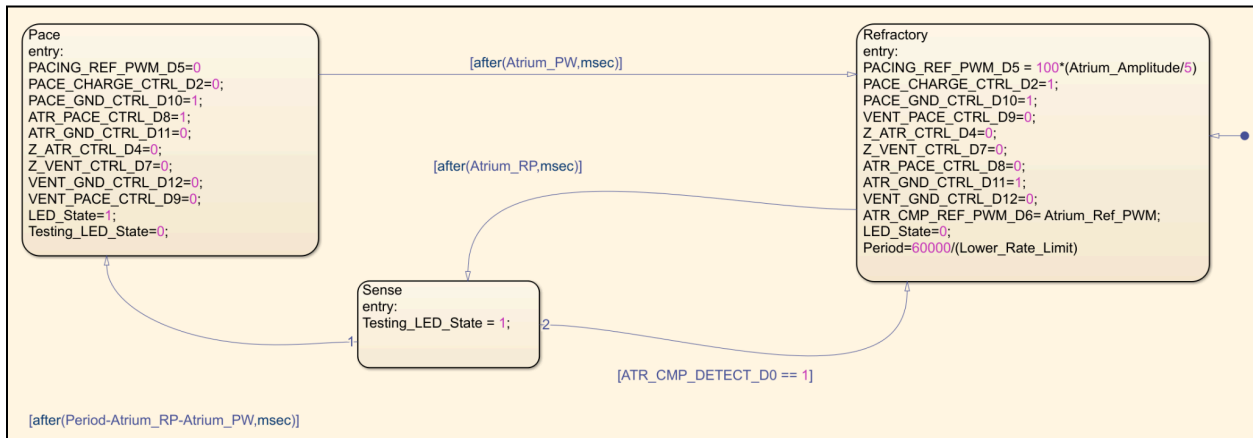


Figure 12: Stateflow diagram of AAI mode on Simulink

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.

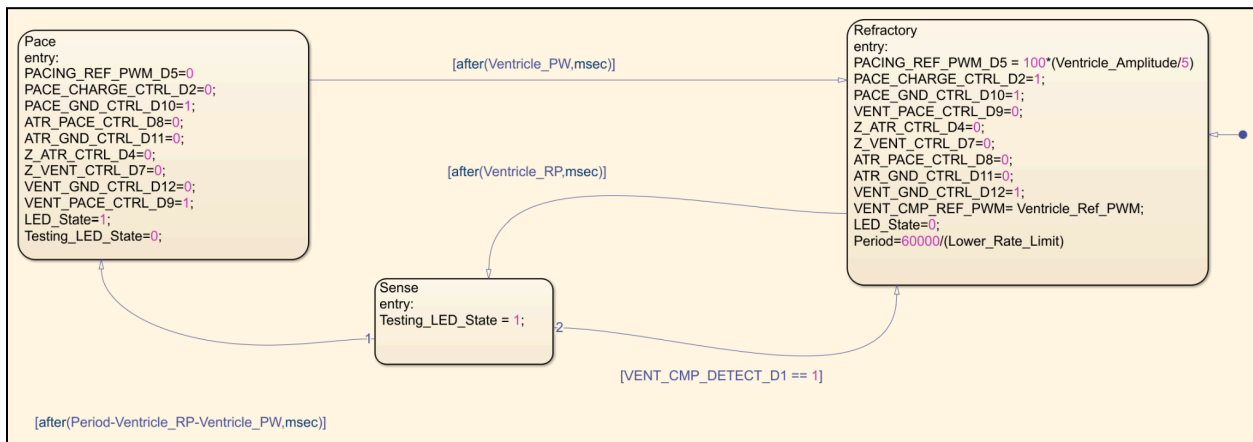


Figure 13: Stateflow diagram of VVI mode on Simulink

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.

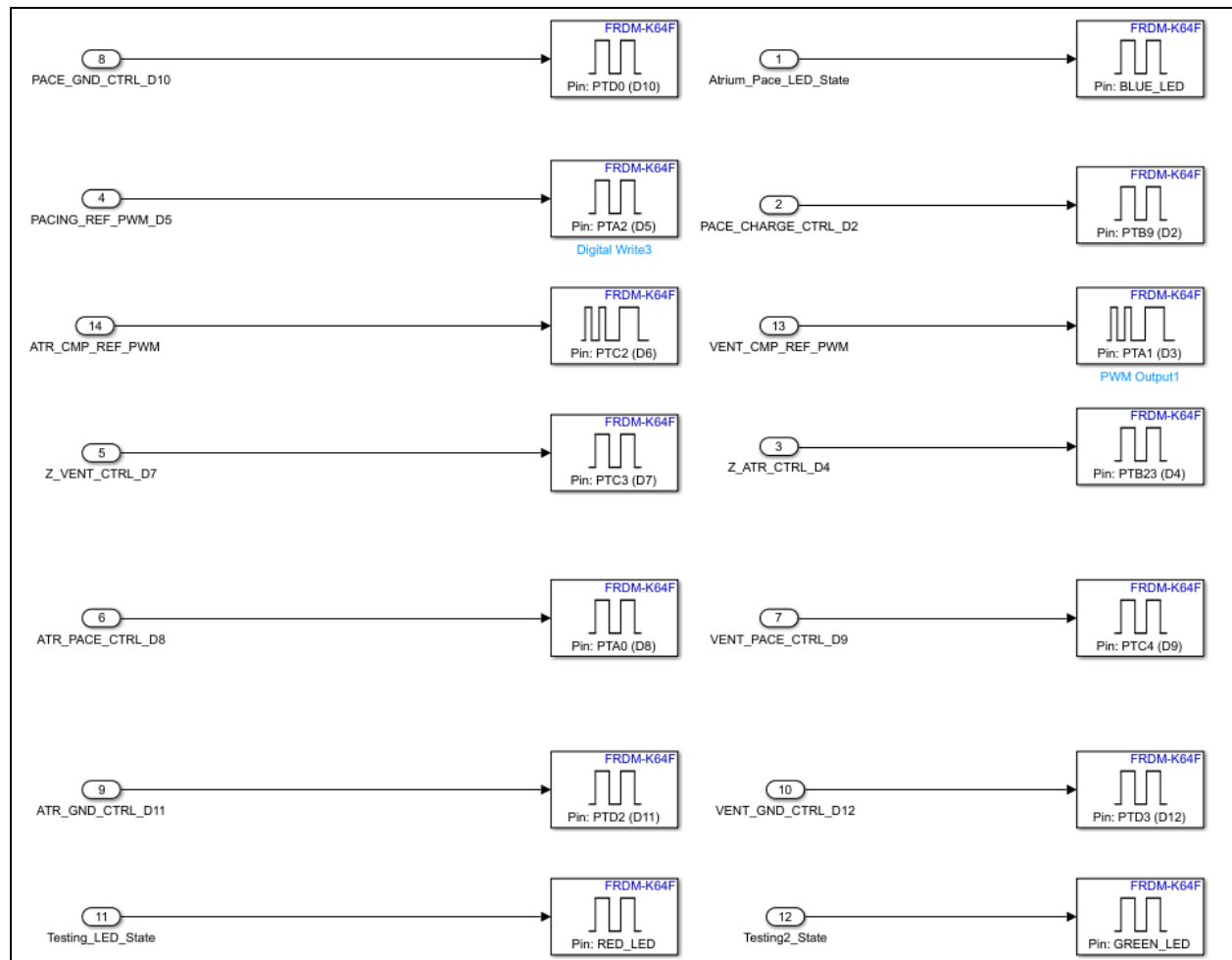


Figure 14: Output parameter subsystem that allows for sensing and pacing.

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 15: 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 16: 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.

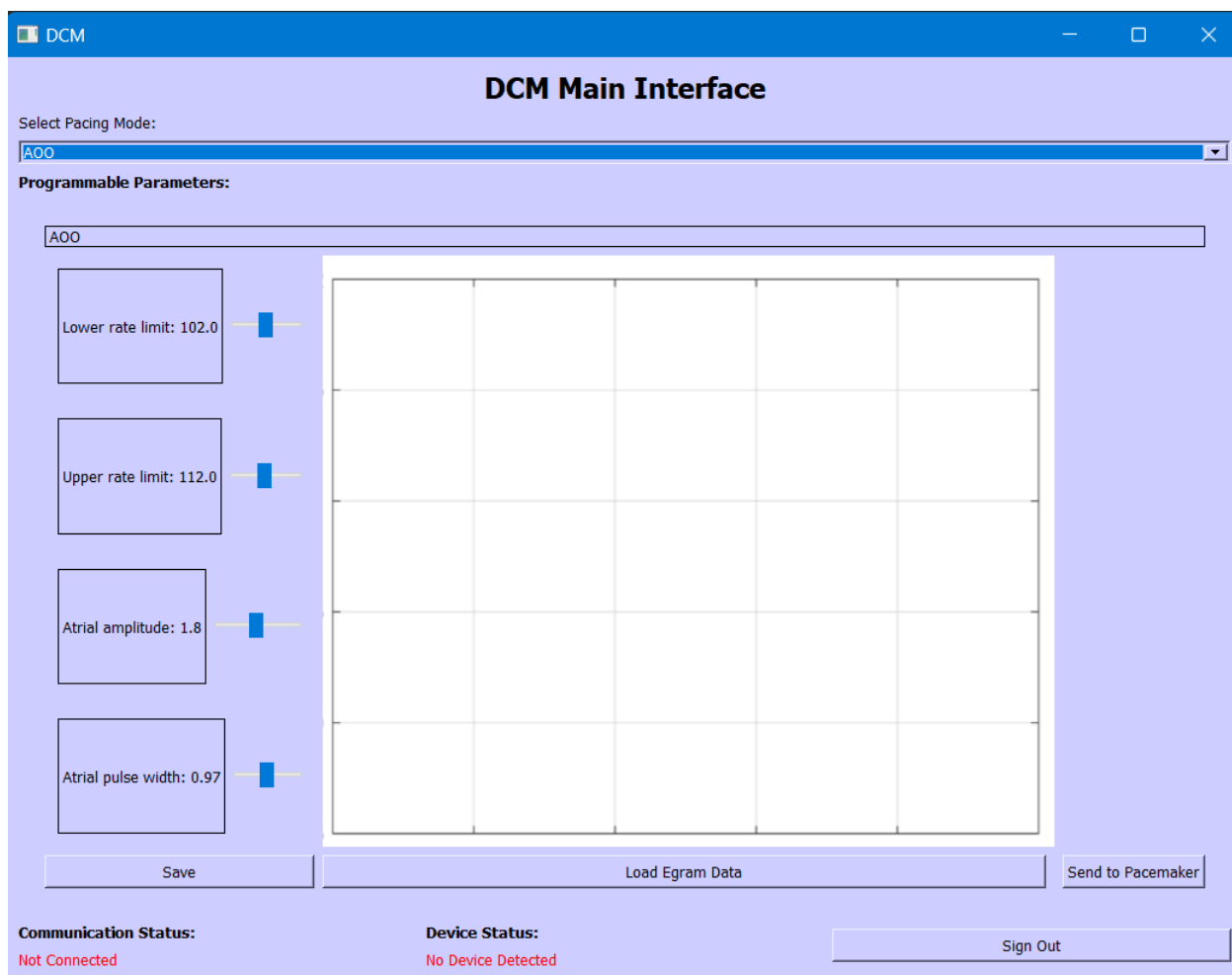


Figure 17: Main Page Set to Mode Select AOO

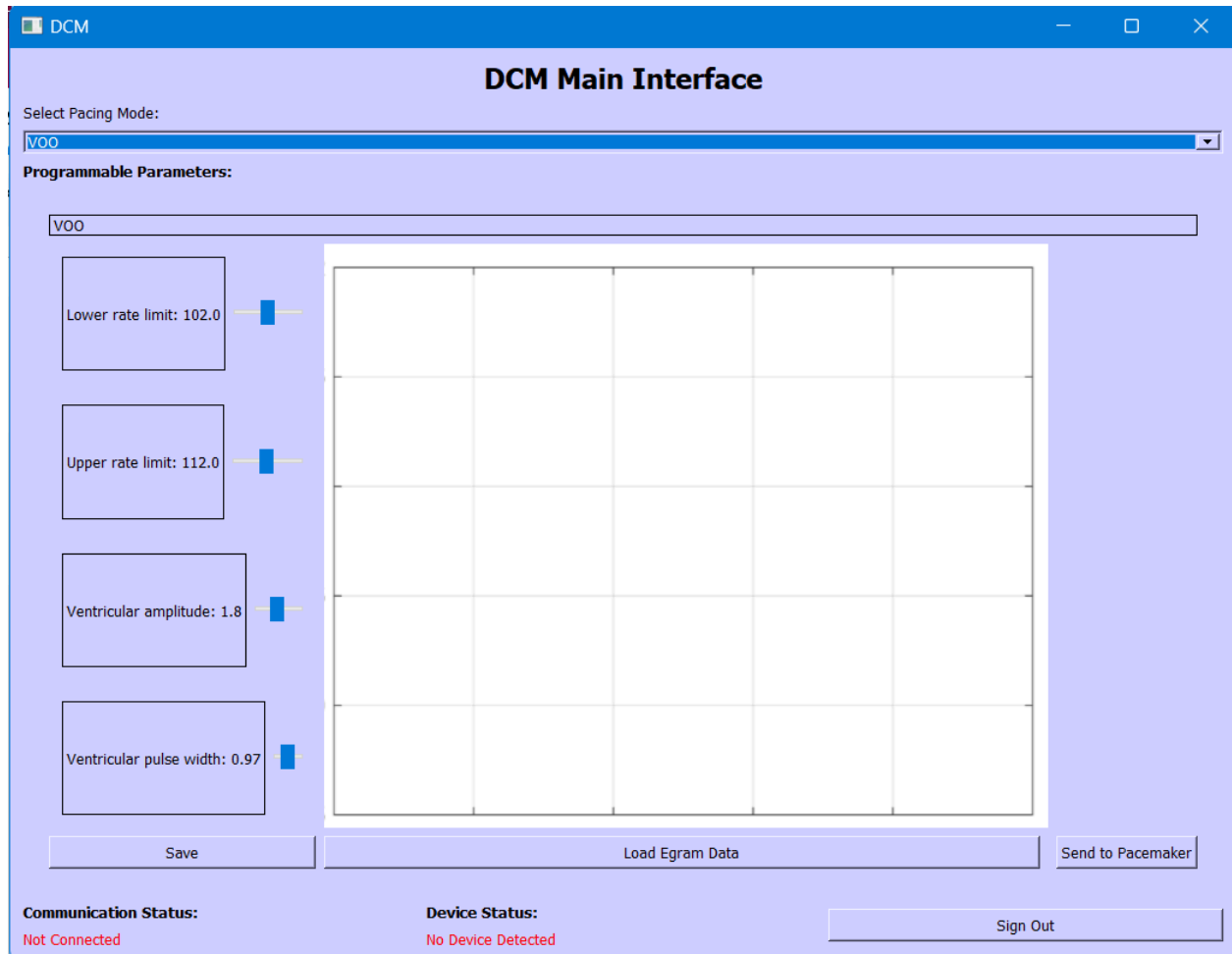


Figure 18: Main Page Set to Mode Select VOO

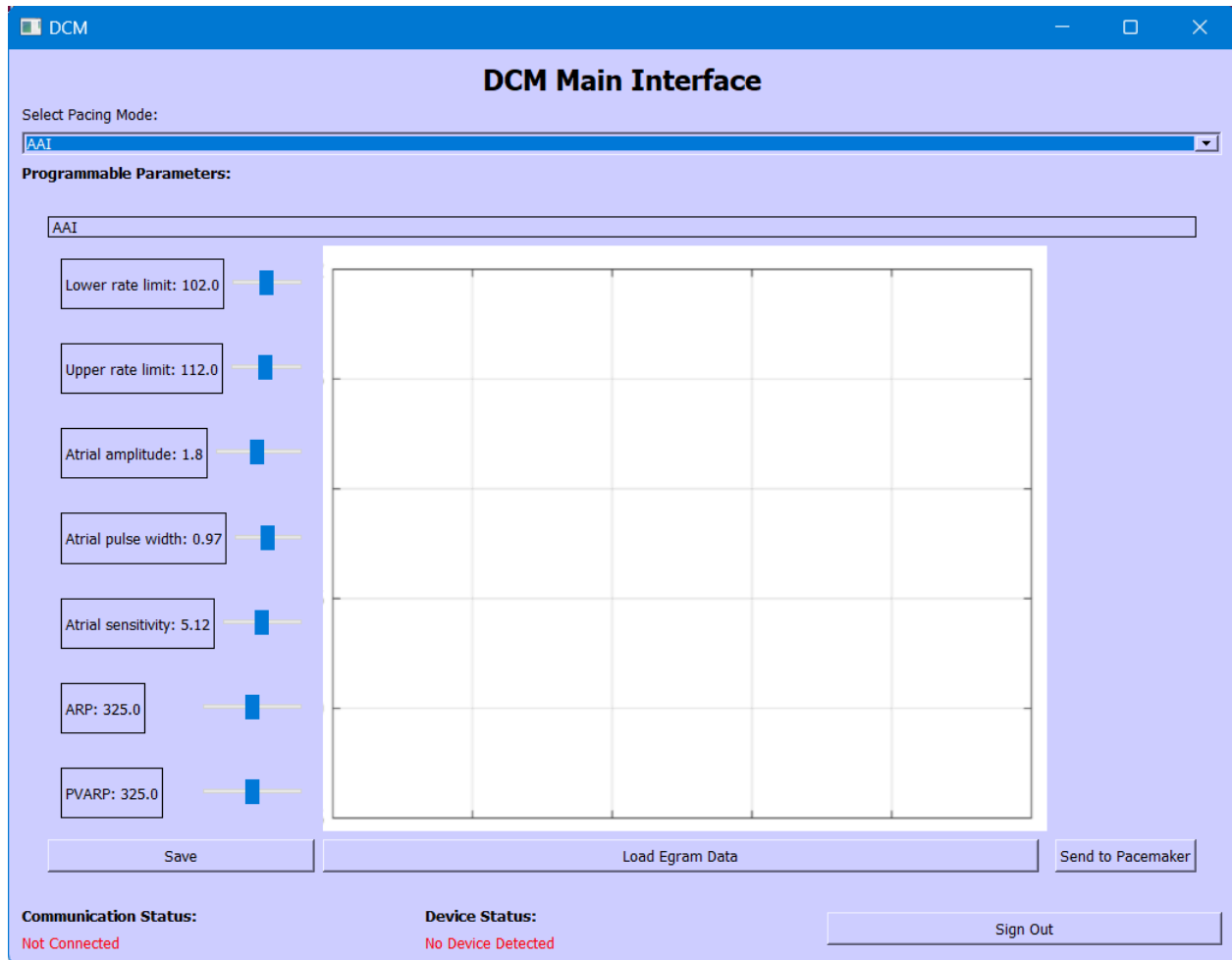


Figure 19: Main Page Set to Mode Select AAI

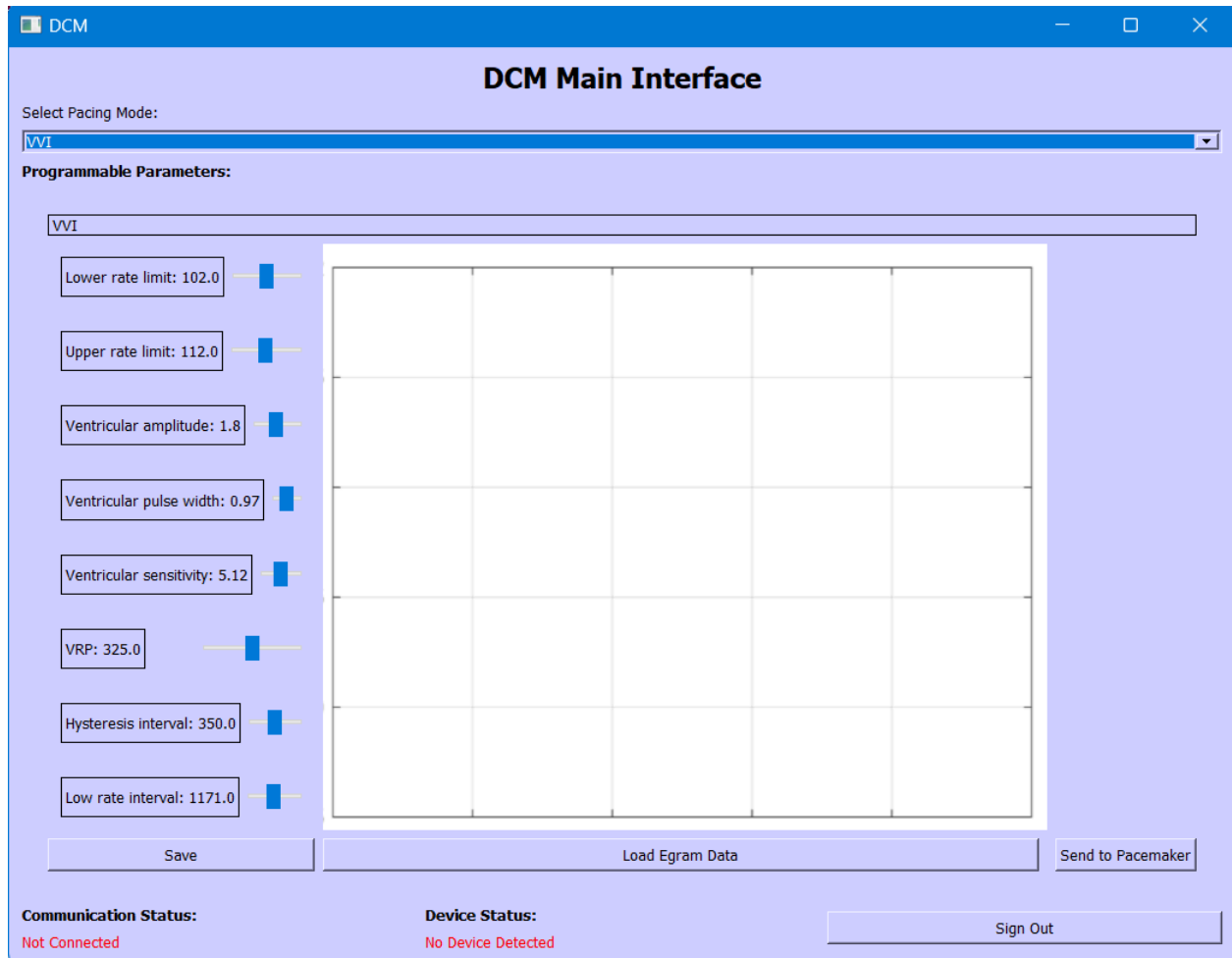


Figure 20: Main Page Set to Mode Select VVI

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

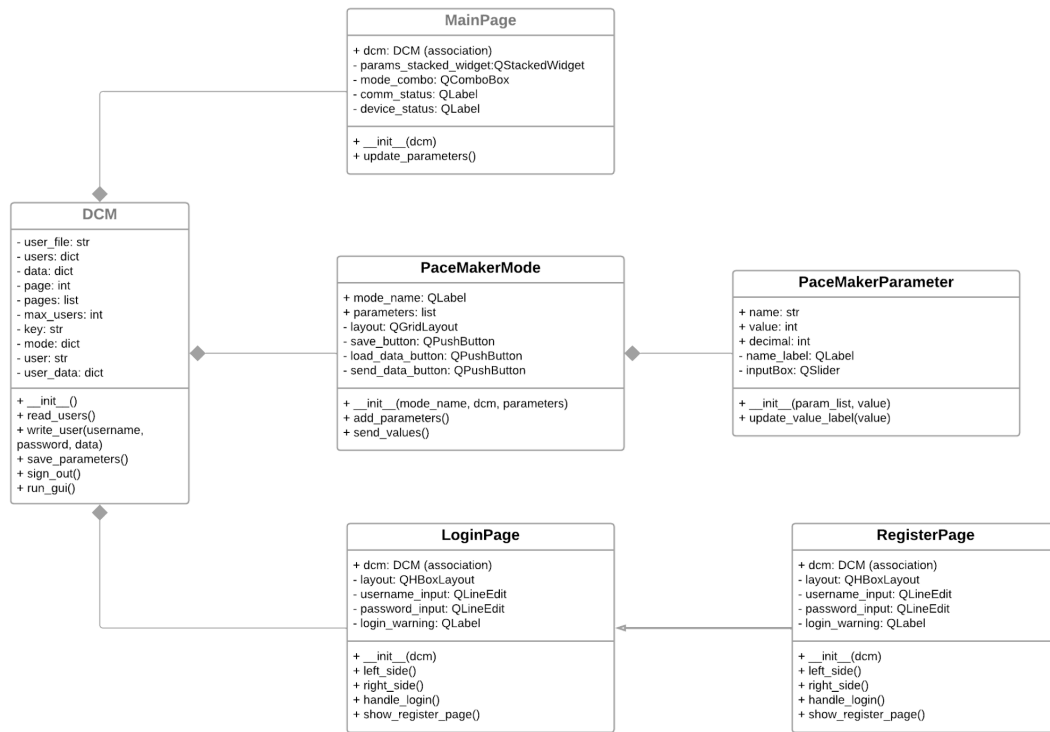


Figure 21: Class diagram for DCM

Classes

Table 4: 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 and orchestrates the primary layout switching between login, registration, and main pages.
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 device.
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.

Methods

Table 5: Method black box behavior

Class	method	Description
DCM	<code>__init__()</code>	<p>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 <code>read_users()</code>. Configures default user-related attributes, including page, pages, max_users, and key, and creates a dictionary of pacing modes (<code>self.mode</code>) and user data (<code>self.user_data</code>). Calls <code>run_gui()</code> to set up the main interface.</p> <p>Args: None.</p> <p>Returns: None.</p> <p>Attributes Modified: Initializes attributes such as <code>self.logo</code>, <code>self.user_file</code>, <code>self.users</code>, <code>self.data</code>, <code>self.page</code>, <code>self.pages</code>, <code>self.max_users</code>, <code>self.key</code>, <code>self.mode</code>, <code>self.user</code>, and <code>self.user_data</code>.</p>

	<code>read_users()</code>	<p>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.</p> <p>Args: None. Returns: None. Attributes Modified: self.users, self.data.</p>
	<code>write_user(username, password, data)</code>	<p>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.</p> <p>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.</p>
	<code>save_parameters()</code>	<p>Saves the current parameters for each pacing mode by retrieving values from self.mode. 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.</p> <p>Args: None. Returns: None. Attributes Modified: None, although it calls write_user() to update user information on file.</p>
	<code>sign_out()</code>	<p>Resets the page attribute to 0, which represents the login screen. Calls run_gui() to reinitialize the interface, showing the login page.</p> <p>Args: None. Returns: None. Attributes Modified: self.page.</p>
	<code>run_gui()</code>	<p>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.</p> <p>Args: None. Returns: None.</p>

		Attributes Modified: self.pages
Mainpage	<code>__init__(dcm)</code>	<p>Initializes the main interface for the pacemaker. Sets up the QStackedWidget to display the selected pacing mode parameters and creates widgets for each mode (AOO, VOO, AAI, VVI). Adds mode selection combo box, communication, and device status labels. Bottom bar layout includes a Sign Out button connected to <code>dcm.sign_out()</code>.</p> <p>Args: dcm (DCM): An instance of the DCM class.</p> <p>Returns: None.</p> <p>Attributes Modified: self.params_stacked_widget, self.mode_combo, self.comm_status, self.device_status.</p>
	<code>update_parameters()</code>	<p>Updates the displayed parameters based on the mode selected in <code>self.mode_combo</code>. It retrieves the selected text, finds the index of this mode in <code>self.mode_combo</code>, and sets <code>self.params_stacked_widget</code> to show the parameters for this index.</p> <p>Args: None.</p> <p>Returns: None.</p> <p>Attributes Modified: None.</p>
PaceMakerMode	<code>__init__(mode_name, dcm, parameters)</code>	<p>Initializes a specific pacing mode, setting up labels and sliders based on provided parameters. Each parameter in <code>parameters</code> is used to create a <code>PaceMakerParameter</code> widget, which is displayed within a grid layout. Sets up "Save," "Load Egram Data," and "Send to Pacemaker" buttons.</p> <p>Args: mode_name (str): Name of the pacing mode. dcm (DCM): DCM instance. parameters (list): List of parameter configurations.</p> <p>Returns: None.</p> <p>Attributes Modified: self.mode_name, self.parameters, self.save_button, self.load_data_button, self.send_data_button.</p>
	<code>add_parameters()</code>	<p>Adds each <code>PaceMakerParameter</code> to the layout for the current pacing mode. It iterates through <code>self.parameters</code> and places each parameter widget in the grid. Sets up additional elements, including a logo and action buttons.</p> <p>Args: None.</p> <p>Returns: None.</p> <p>Attributes Modified: None.</p>
	<code>send_values()</code>	Collects the current values of all parameters in the

		<p>mode by calling value for each PaceMakerParameter in self.parameters. Returns a list of these values.</p> <p>Args: None.</p> <p>Returns: list: List of current parameter values.</p> <p>Attributes Modified: None.</p>
PaceMakerParameter	<code>__init__(param_list, value)</code>	<p>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.</p> <p>Args: param_list (list): List of parameter details (name, min, max, decimal). value (int): Initial value of the parameter.</p> <p>Returns: None.</p> <p>Attributes Modified: self.name, self.value, self.decimal, self.name_label, self.inputBox.</p>
	<code>update_value_label(value)</code>	<p>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.</p> <p>Args: value (int): Current slider value.</p> <p>Returns: None.</p> <p>Attributes Modified: self.value.</p>
LoginPage	<code>__init__(dcm)</code>	<p>Initializes the login page, setting up the main layout, which includes the left_side (login form and buttons) and right_side (logo display).</p> <p>Args: dcm (DCM): The main DCM instance for managing application state.</p> <p>Returns: None.</p> <p>Attributes Modified: self.layout, self.left_side_layout, self.right_side_layout.</p>
	<code>left_side()</code>	<p>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.</p> <p>Args: None.</p> <p>Returns: None.</p> <p>Attributes Modified: self.username_input, self.password_input, self.login_warning.</p>
	<code>right_side()</code>	<p>Configures the right side of the login page with a logo, which is centered in self.right_side_layout.</p> <p>Args: None.</p>

		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.

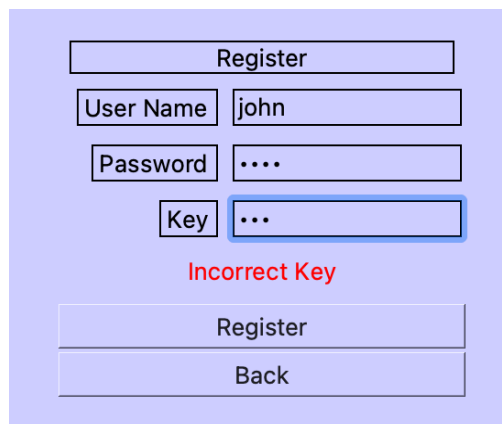
Testing Results - DCM

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 6: Incorrect key

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



The screenshot shows a registration form with the following elements:

- A "Register" button at the top.
- Input fields for "User Name" (containing "john"), "Password" (containing "...."), and "Key" (containing "...").
- A red error message "Incorrect Key" displayed below the "Key" field.
- Buttons for "Register" and "Back" at the bottom.

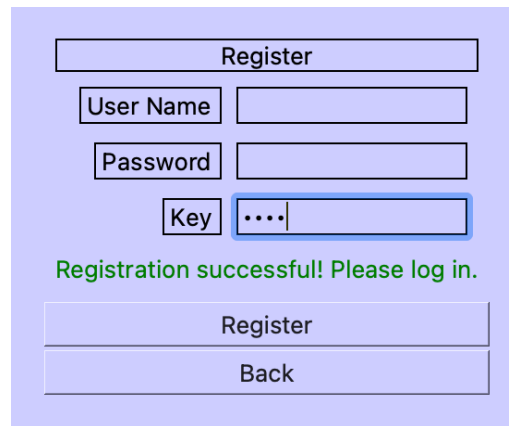
Figure 22: Registration Case 1

Registration - Case 2:

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

Table 7: correct key

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



The screenshot shows a registration form with a light blue background. At the top is a button labeled "Register". Below it are three input fields: "User Name", "Password", and "Key". The "Key" field contains four dots. Below the input fields, the text "Registration successful! Please log in." is displayed in green. At the bottom are two buttons: "Register" and "Back".

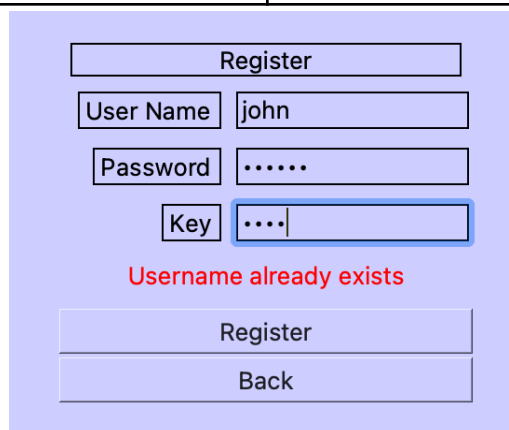
Figure 23: Registration Case 2

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 8: Existing username

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



The screenshot shows the same registration form as Figure 23. The "User Name" field now contains the text "john". The "Key" field still contains four dots. Below the input fields, the text "Username already exists" is displayed in red. The "Register" and "Back" buttons are still at the bottom.

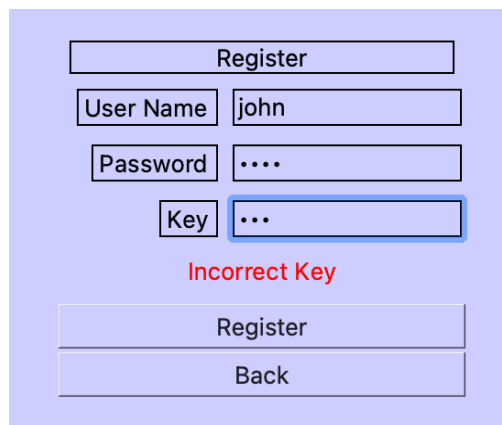
Figure 24: Registration Case 3

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 9: Incorrect key & existing username

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



The screenshot shows a registration form with the following elements:

- A "Register" button at the top.
- Input fields for "User Name" (containing "john"), "Password" (containing "...."), and "Key" (containing "...").
- A red error message "Incorrect Key" displayed below the "Key" field.
- Buttons for "Register" and "Back" at the bottom.

Figure 25: 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 10: Maximum users

Case	Input	Expected Output	Output
New username Correct key 11th Account	Username: new_user10 Password: snow Key: 123	Maximum Users Reached	Maximum Users Reached

Register

User Name new_user10

Password ●●●●

Key ●●●●

Maximum users reached

Register

Back

Figure 26: 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 11: Incorrect password

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

Login

User Name john

Password ●●●●●●

Incorrect Username or Password

Login

Register

Figure 27: 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 username or password is incorrect and the login is unsuccessful

Table 12: Incorrect username

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

The screenshot shows a login interface with a light blue background. At the top is a button labeled 'Login'. Below it are two input fields: 'User Name' containing the text 'jonny' and 'Password' containing four dots. Below the password field, the text 'Incorrect Username or Password' is displayed in red. At the bottom are two buttons: 'Login' and 'Register'.

Figure 28: 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 13: Maximum users

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

The screenshot shows a login interface with a light blue background. At the top is a button labeled 'Login'. Below it are two input fields: 'User Name' containing the text 'jonny' and 'Password' containing eight dots. Below the password field, the text 'Incorrect Username or Password' is displayed in red. At the bottom are two buttons: 'Login' and 'Register'.

Figure 29: Login Case 3

Main Page Test Cases:

Sign Out Button:

Initial:

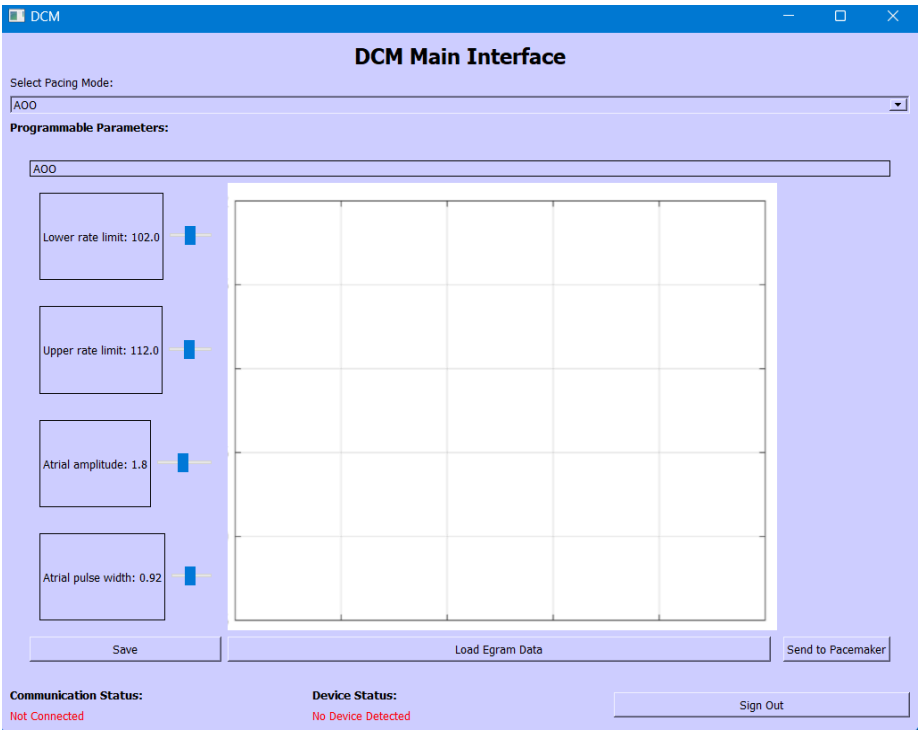


Figure 30: DCM Interface Before Clicking Signout

After:

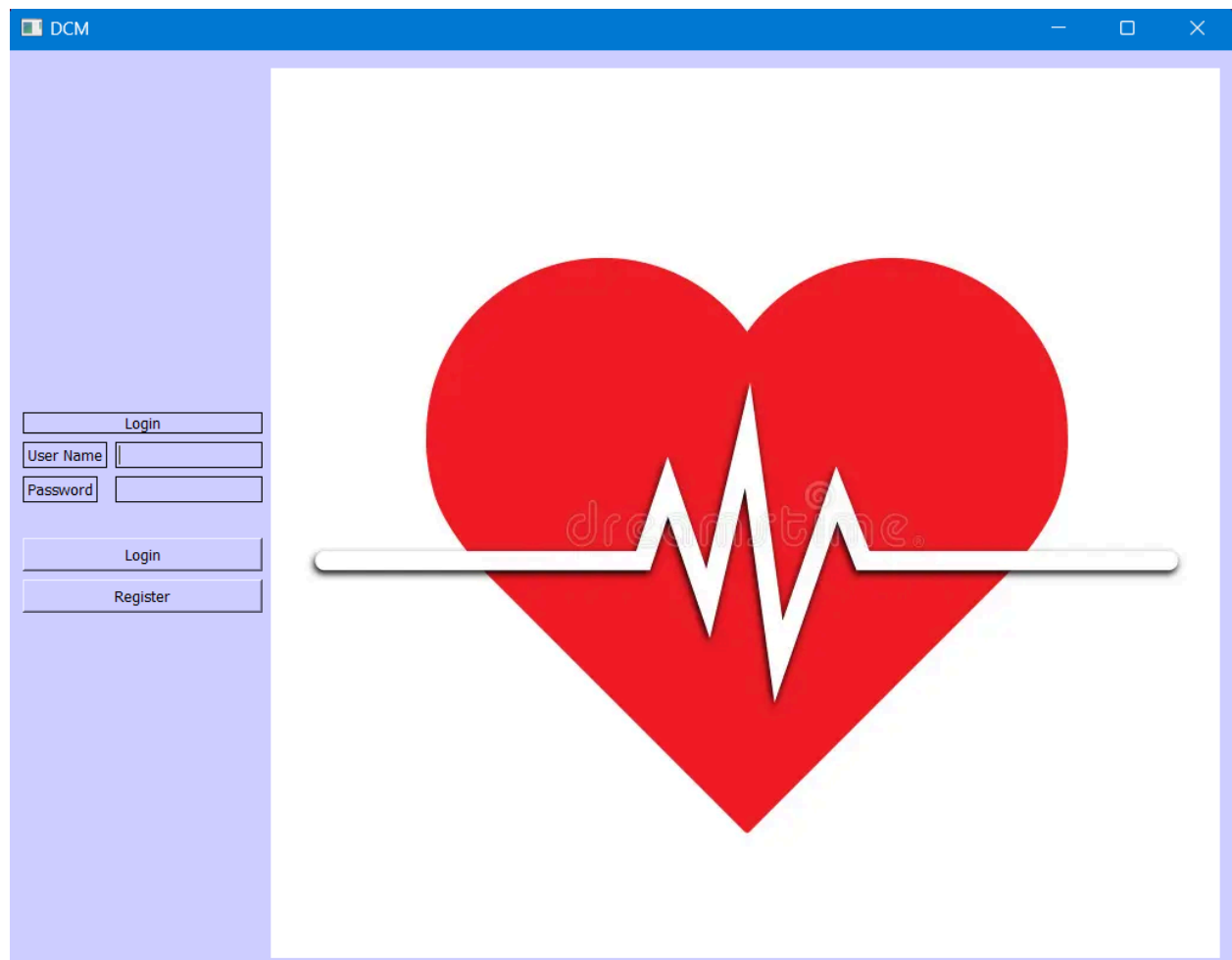


Figure 31: DCM Interface After Clicking Signout

Checking Saved Parameters:

When Switching Modes:

Initial:

The screenshot displays the 'DCM Main Interface' with a light blue background. At the top, a 'Select Pacing Mode:' dropdown menu is set to 'AOO'. Below this, the 'Programmable Parameters:' section contains a sub-header 'AOO' and a large empty grid for ECG data. On the left side of the grid, four parameter boxes are visible, each with a blue slider: 'Lower rate limit: 102.0', 'Upper rate limit: 112.0', 'Atrial amplitude: 1.8', and 'Atrial pulse width: 0.97'. At the bottom of the interface, there are three buttons: 'Save', 'Load Egram Data', and 'Send to Pacemaker'. The status section at the very bottom shows 'Communication Status: Not Connected' in red, 'Device Status: No Device Detected' in red, and a 'Sign Out' button.

DCM Main Interface

Select Pacing Mode: AOO

Programmable Parameters:

AOO

Lower rate limit: 102.0

Upper rate limit: 112.0

Atrial amplitude: 1.8

Atrial pulse width: 0.97

Save Load Egram Data Send to Pacemaker

Communication Status:
Not Connected

Device Status:
No Device Detected

Sign Out

Figure 32: DCM While Setting Up Mode AOO Parameters

Switching:

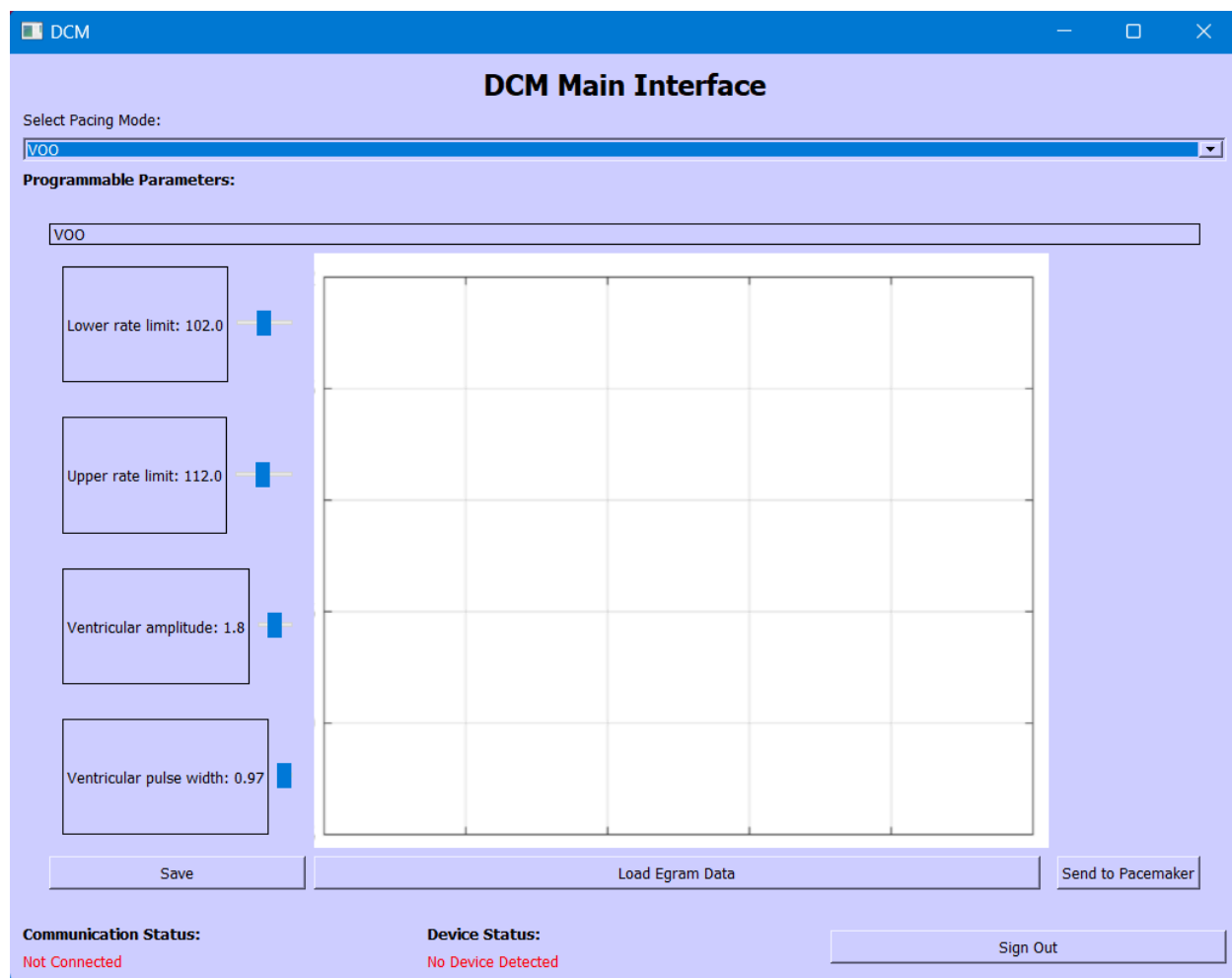


Figure 33: DCM While Setting Up Mode VOO Parameters

After:

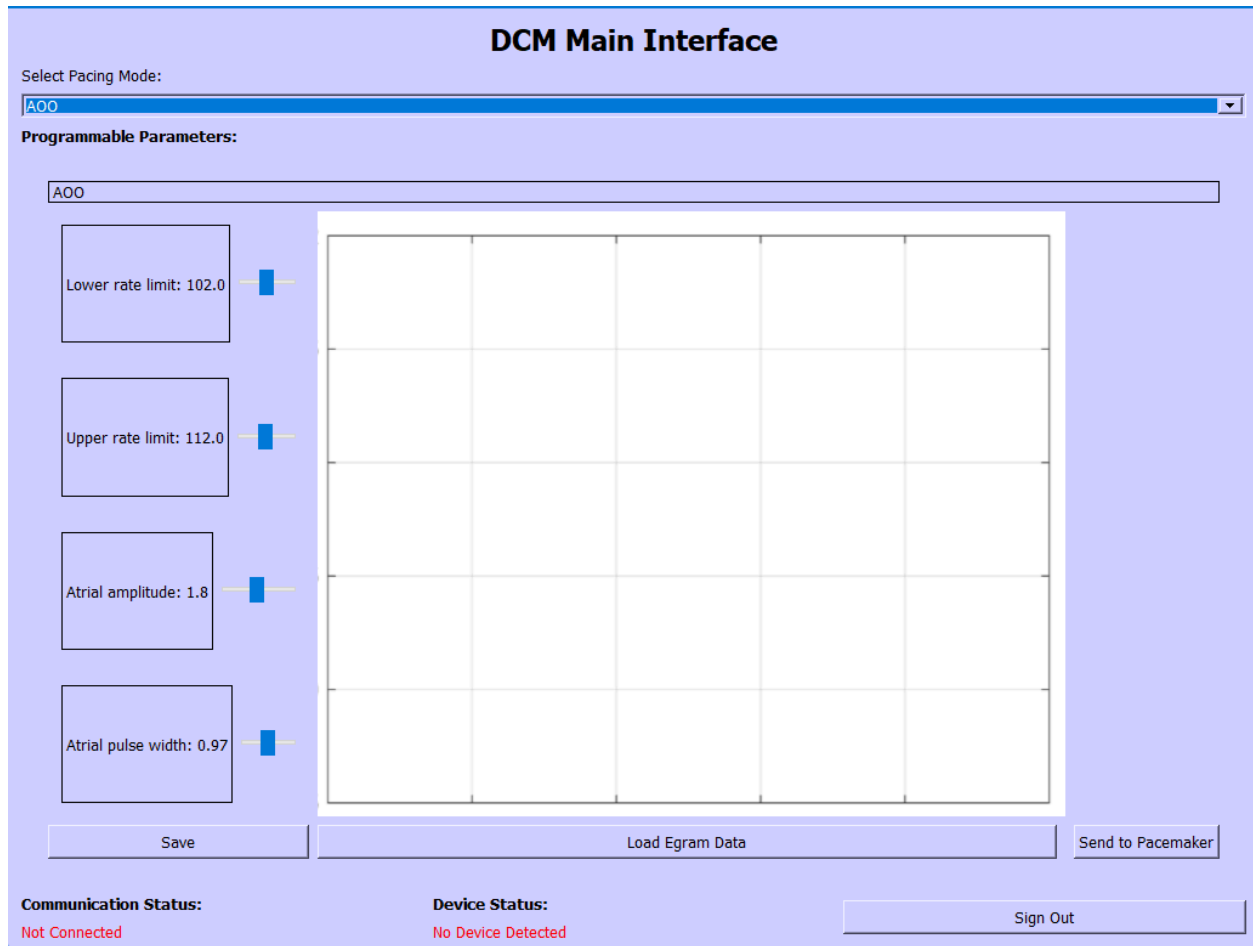


Figure 34: DCM While Setting Up Mode AOO Parameters. The sliders and the values stayed the same as expected.

When signing out and signing back in:

Initial:

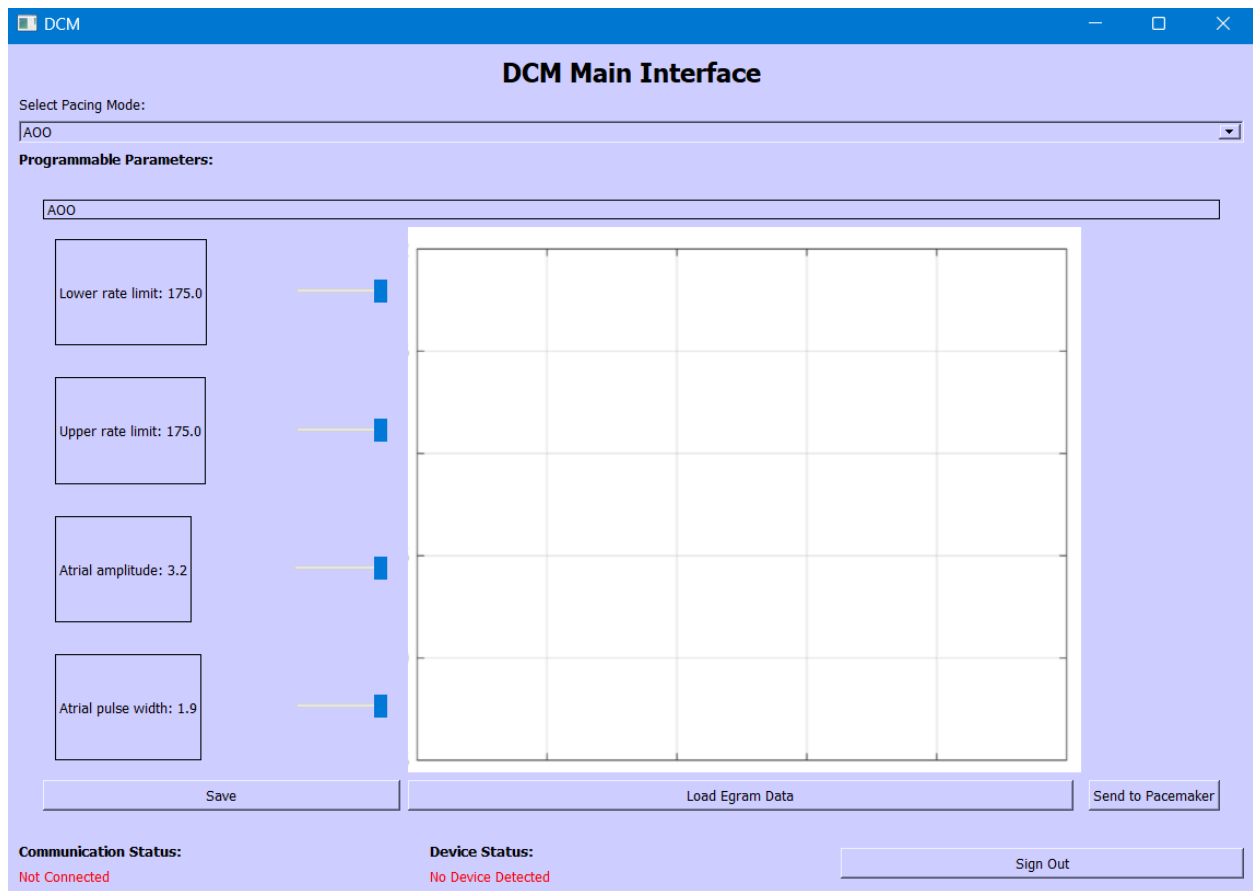


Figure 35: DCM While Setting Up Mode AOO Parameters

*Save is Pressed Here

Switching:



Figure 36: DCM While Signed Out

*Logged In with same User

After:

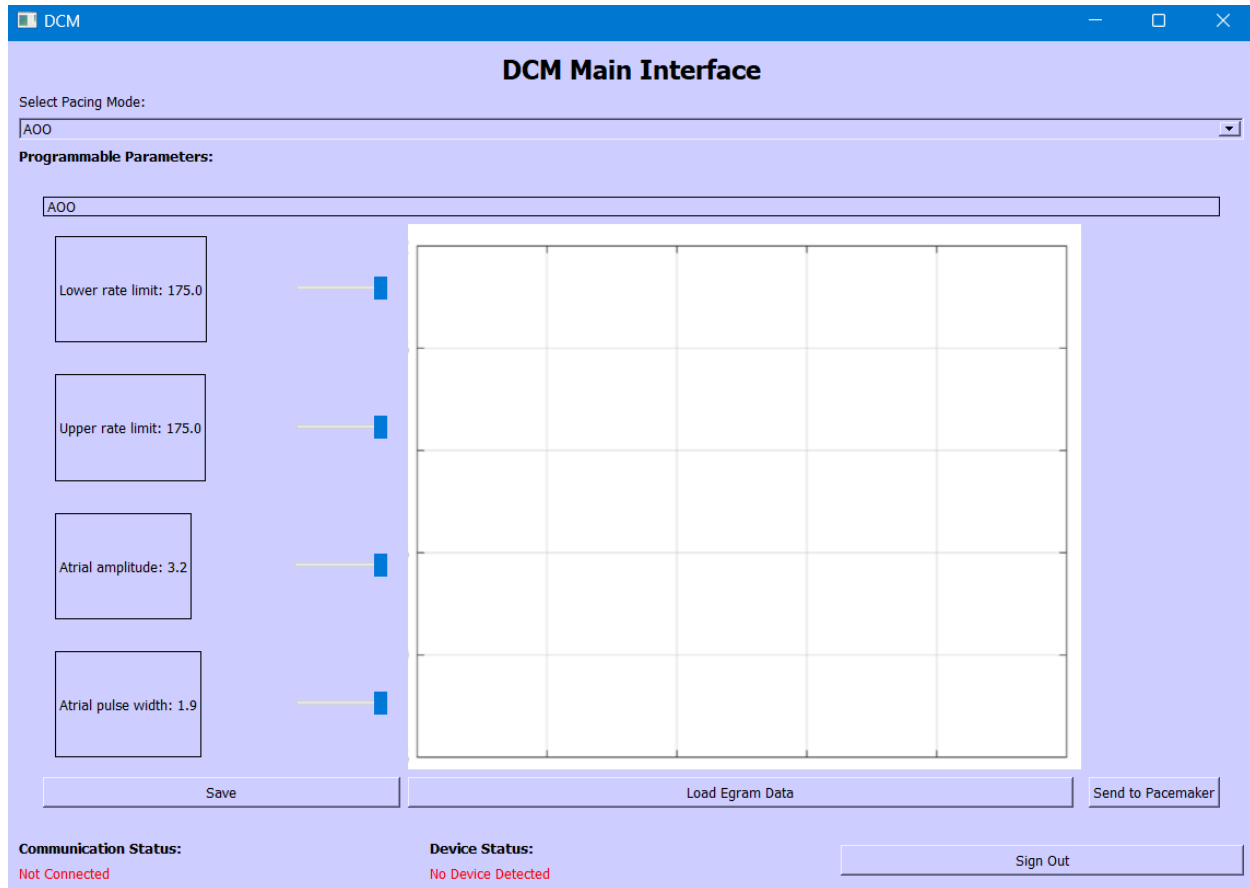


Figure 37: DCM While Logged In

*Display comes Like this

Problems and Challenges:

DCM

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.

Simulink

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.

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

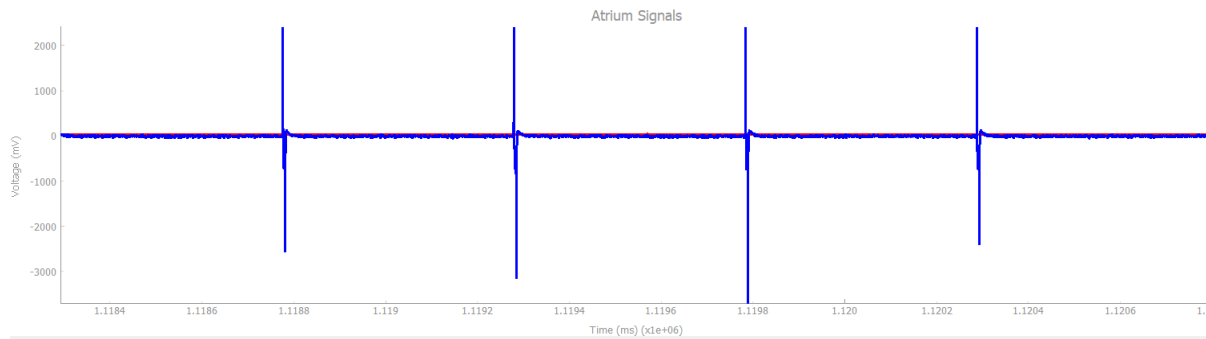


Figure 38: AAO Test Case 1 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = OFF

AOO - Case 2

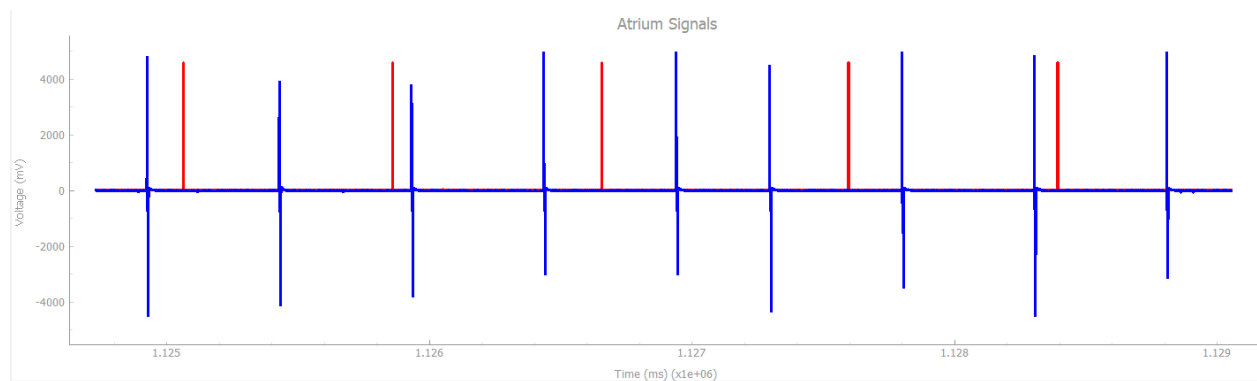


Figure 39: AAO Test Case 2 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = ON, BPM = 75

VOO - Case 1

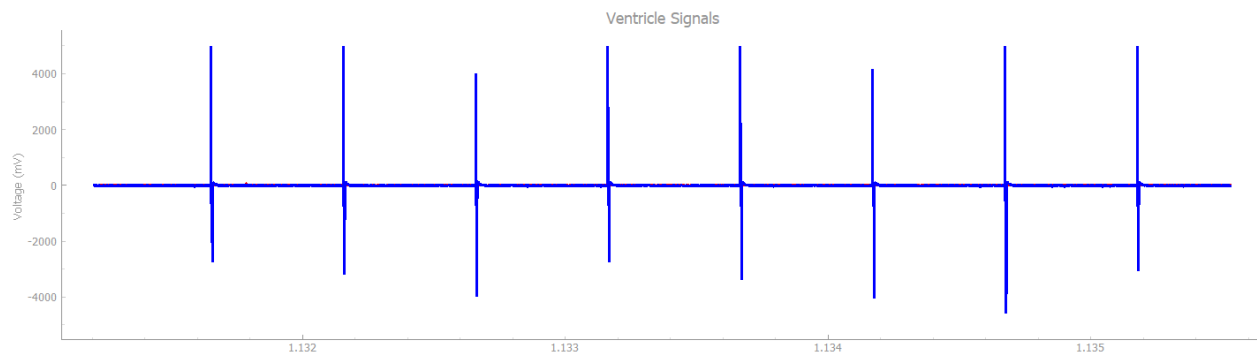


Figure 40: VOO Test Case 1 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = OFF

VOO - Case 2

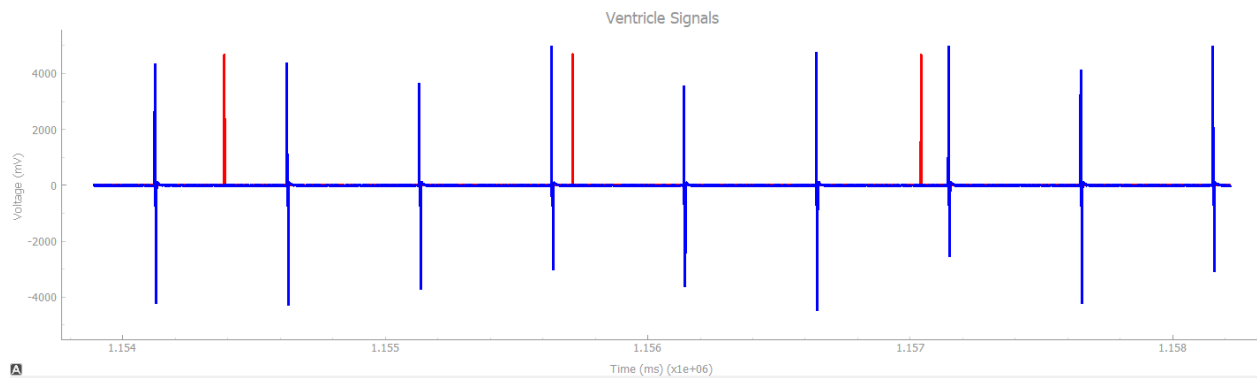


Figure 41: VOO Test Case 2 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 45

AAI - Case 1

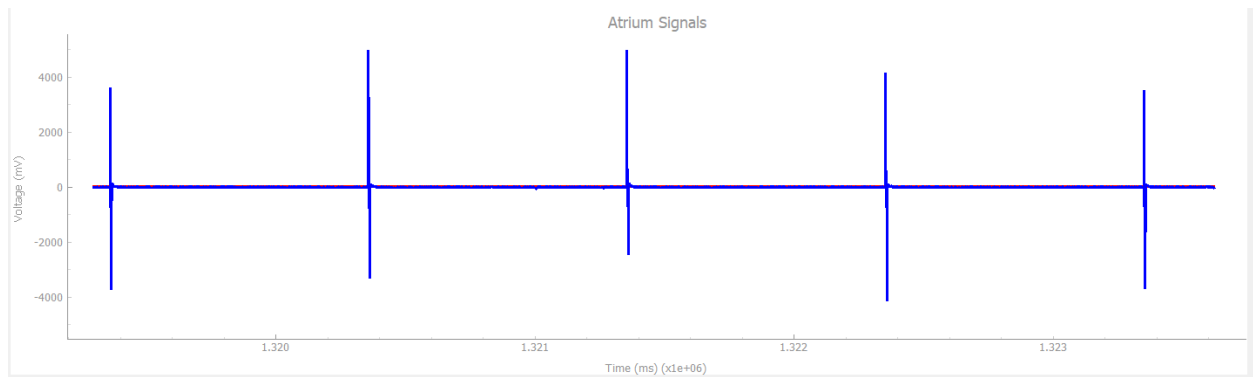


Figure 42: AAI Test Case 1 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = OFF

AAI - Case 2

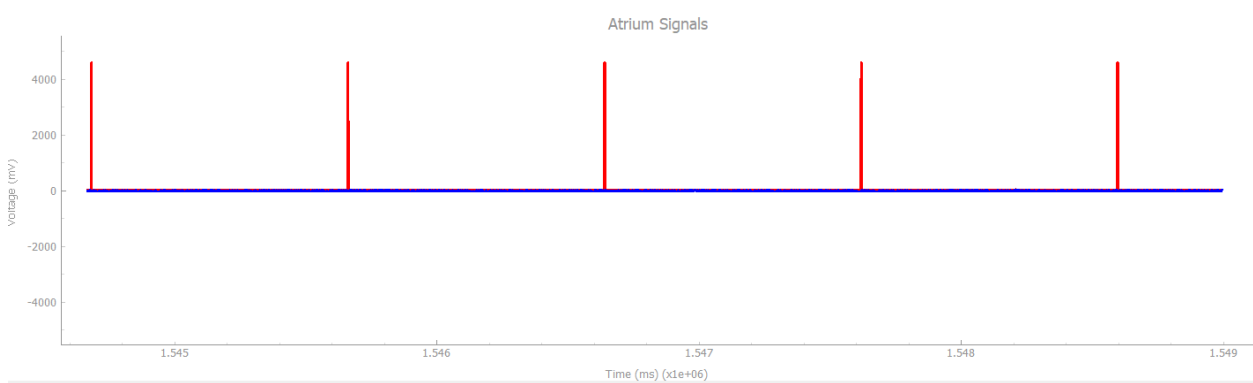


Figure 43: AAI Test Case 2 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = ON, BPM = 61

AAI - Case 3

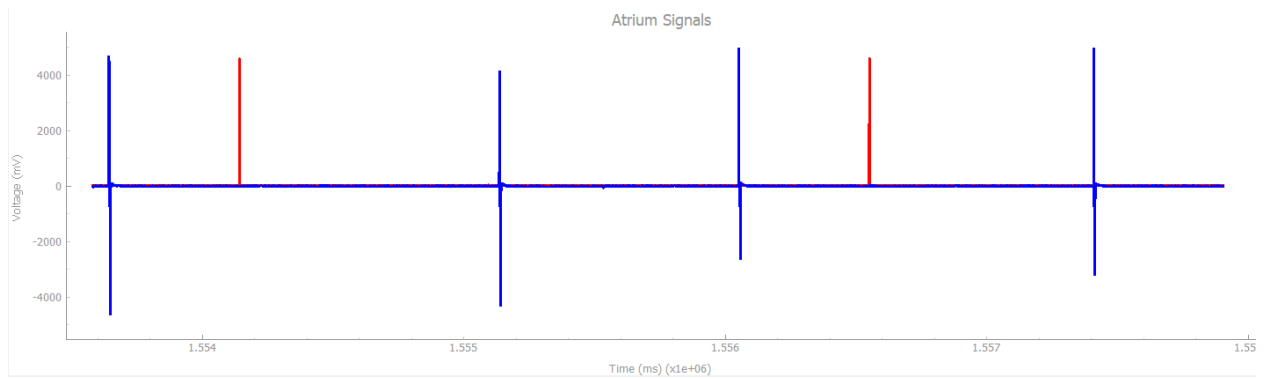


Figure 44: AAI Test Case 3 - Atrial Amp = 5, LRL = 60, Atrial Pulse Width = 5, Natural Atrium = ON, BPM = 40

VVI - Case 1

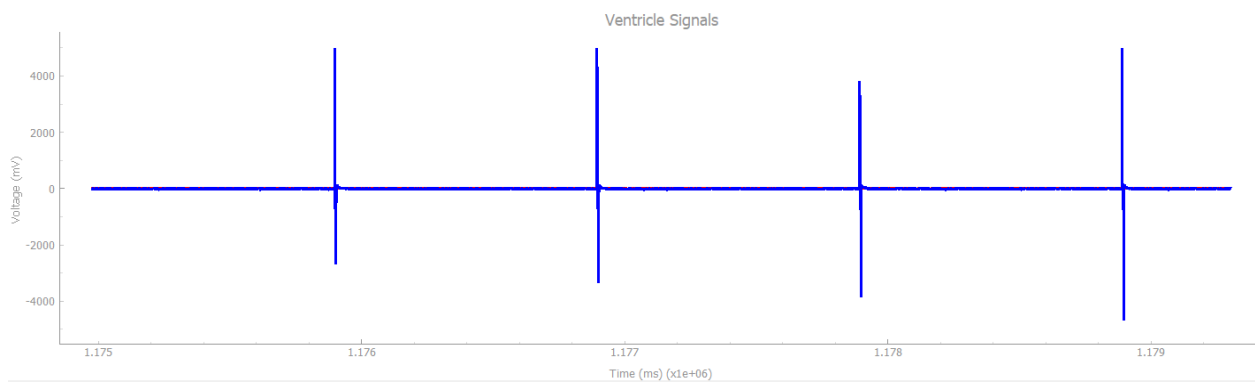


Figure 45: VVI Test Case 1 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = OFF

VVI - Case 2

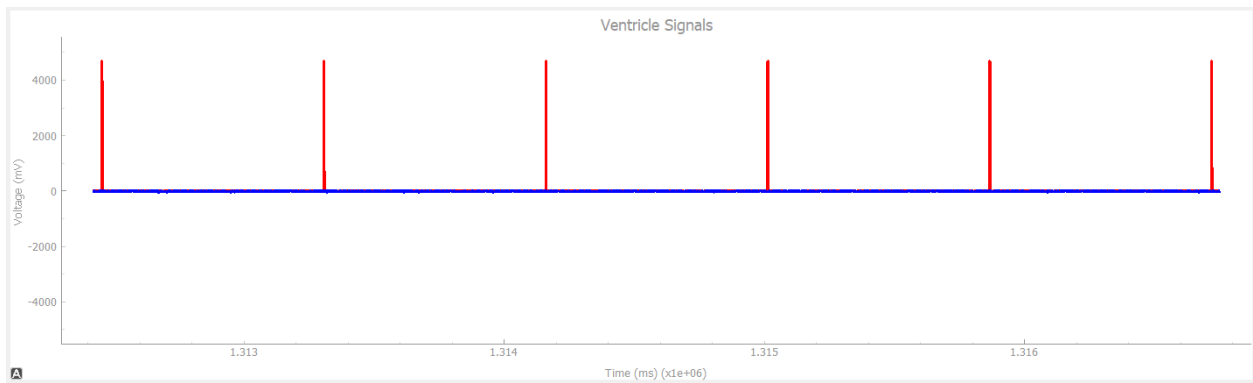


Figure 46: VVI Test Case 2 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 70

VVI - Case 3

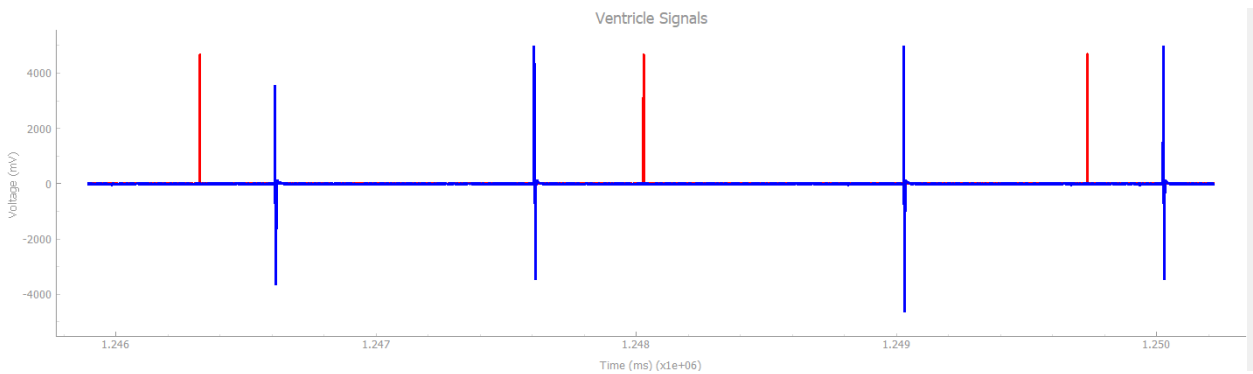


Figure 47: VVI Test Case 3 - Ventricle Amp = 5, LRL = 60, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 35

Additional Tests

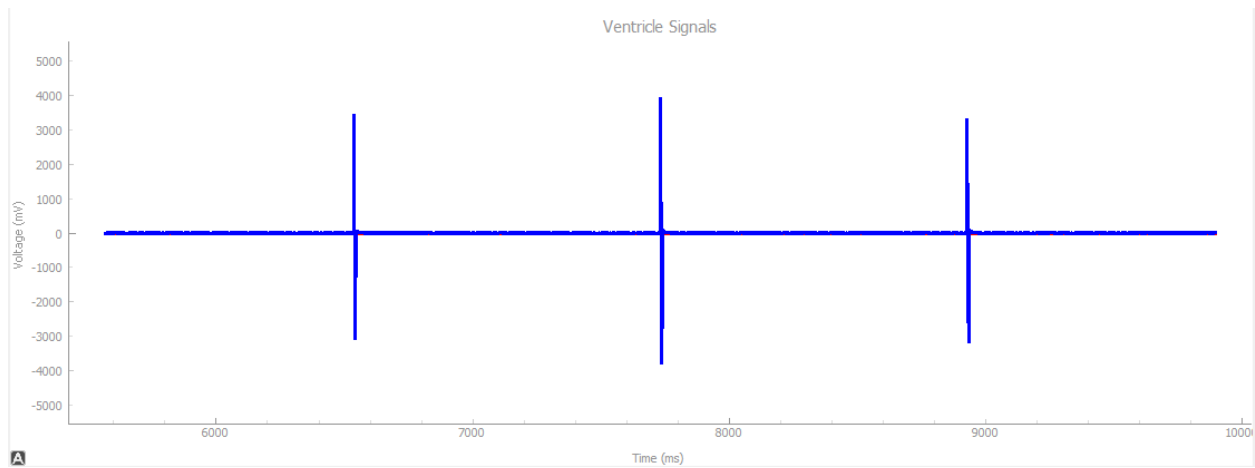


Figure 48: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = OFF

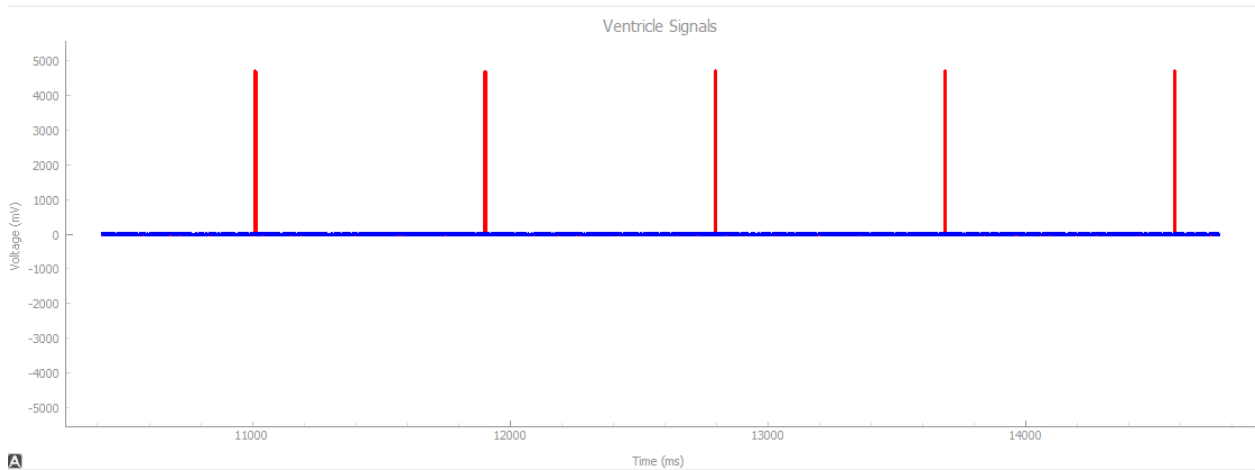


Figure 49: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 67 BPM

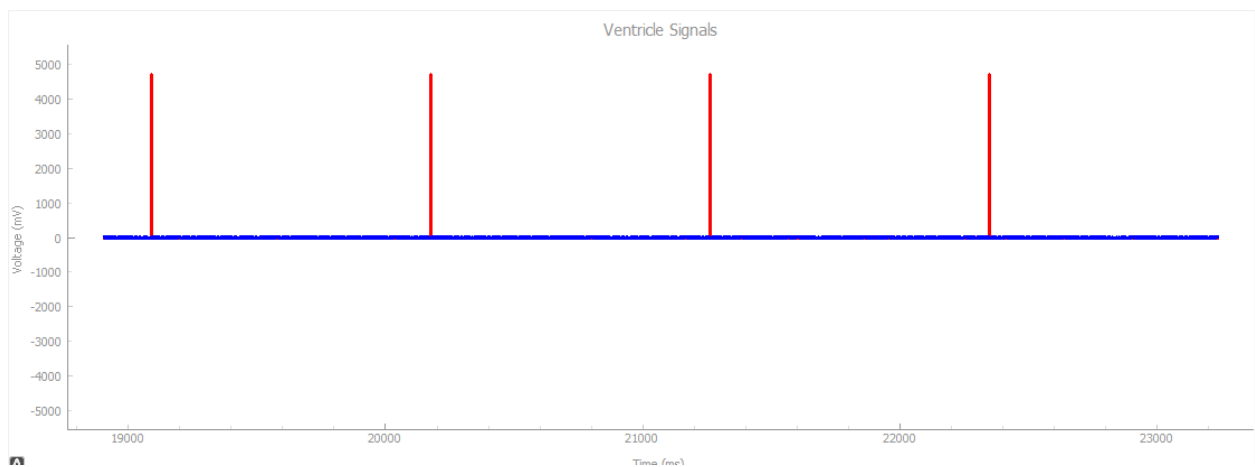


Figure 50: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 55 BPM

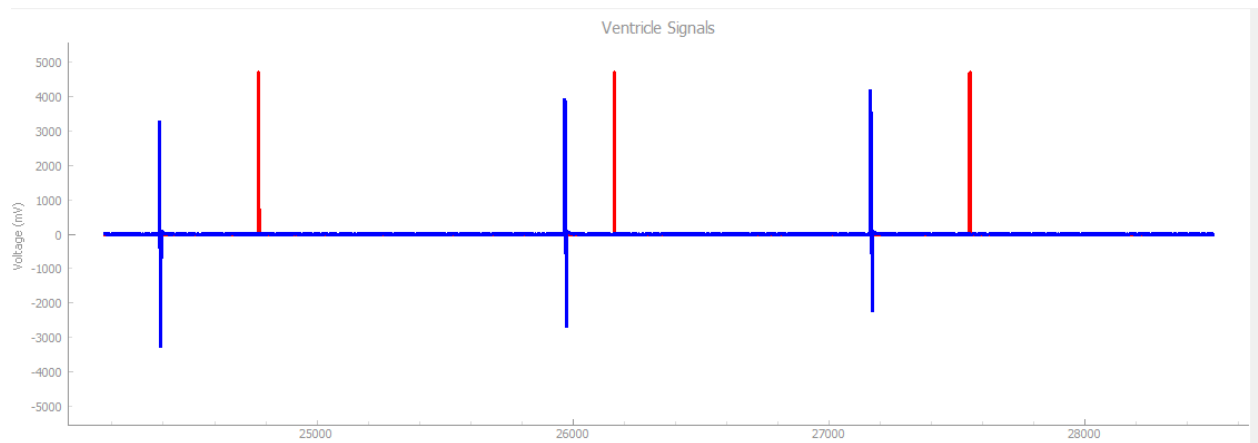


Figure 51: Mode - VVI, Ventricle Amp = 5, LRL = 50, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 43 BPM

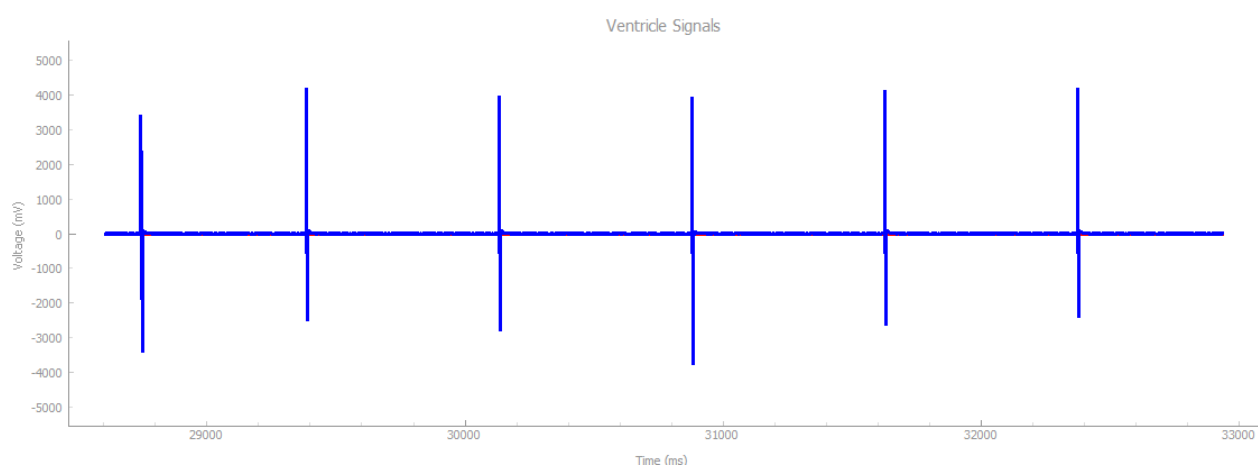


Figure 52: Mode - VVI, Ventricle Amp = 5, LRL = 80, Ventricle Pulse Width = 5, Natural Ventricle = OFF

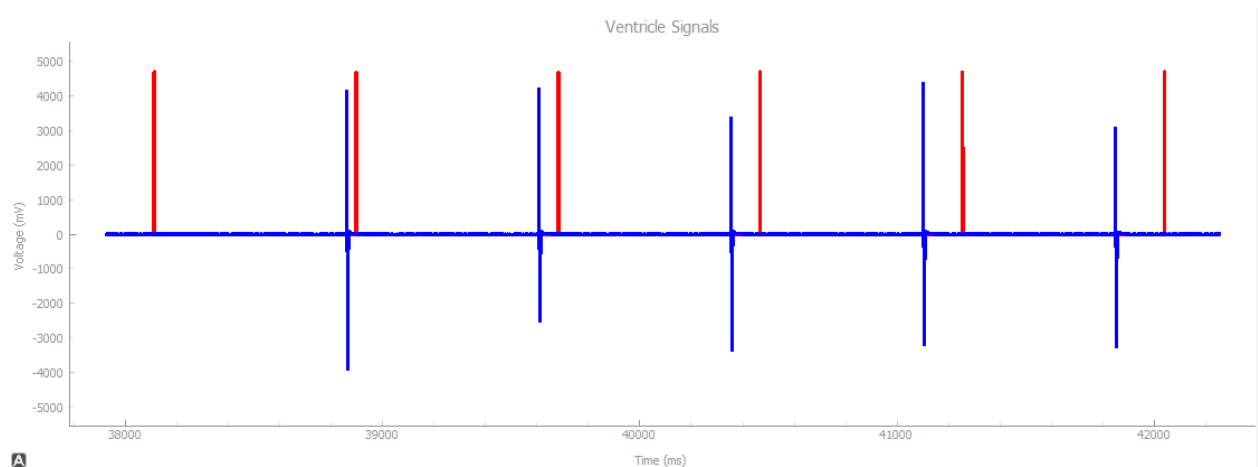


Figure 53: Mode - VVI, Ventricle Amp = 5, LRL = 80, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 76 BPM

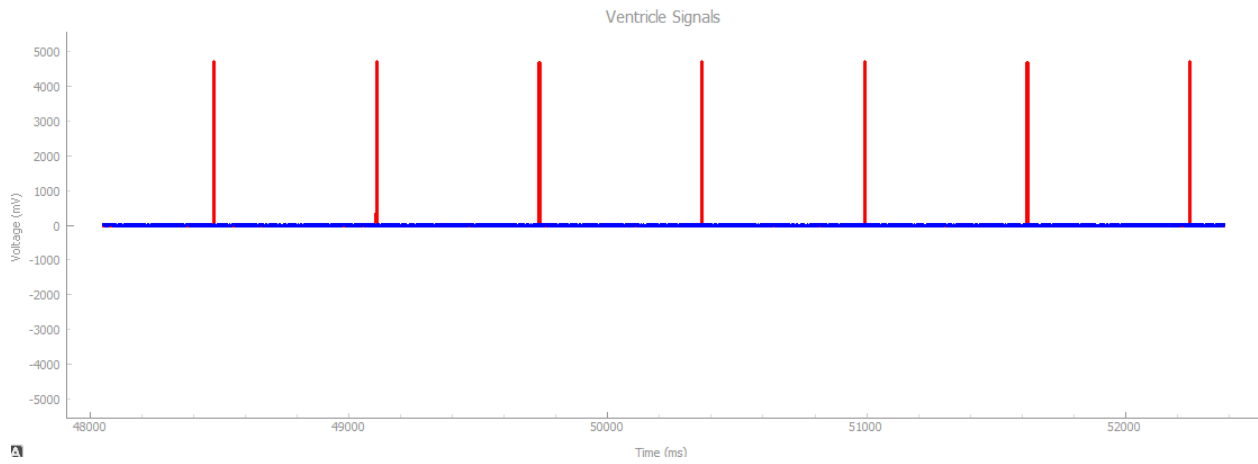


Figure 54: Mode - VVI, Ventricle Amp = 5, LRL = 80, Ventricle Pulse Width = 5, Natural Ventricle = ON, BPM = 95 BPM

Future Project Changes

DCM

- 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 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.

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

Simulink

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.

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).