

# Assignment 1 – Pacemaker Project

SFWRENG 3K04 – Software Development

James Cameron – camerj22 – 400450866

Ryan Brubacher – brubachr - 400457266

Anuja Perera - perera5 - 400459978

Matthew Rakic – rakicm1 - 400449728

Jack Vu – vuj23 - 400453031

Hunter Boyd – boydh4 - 400384654

## ACADEMIC STATEMENT

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is our own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario.

October 25, 2024

Lab L01

## Table of Contents

<b>1. Introduction.....</b>	<b>4</b>
<b>2. Pacemaker.....</b>	<b>6</b>
2.1 The Pacemaker.....	6
2.2 Requirements.....	6
2.3 Design Decisions .....	7
2.4 Simulink Model Overview .....	9
2.4.1 Main Simulink Model .....	9
2.4.2 Programmable Parameters Subsystem .....	10
2.4.3 Sensing Subsystem .....	11
2.4.4 Microcontroller Pin Mapping Subsystem .....	12
2.4.5 Main Stateflow Chart.....	13
2.4.6 VOO Stateflow.....	14
2.4.7 AOO Stateflow.....	15
2.4.8 VVI Stateflow.....	16
2.4.9 AAI Stateflow.....	17
2.5 Testing and Results .....	18
2.6 Potential Changes and Future Considerations.....	27
2.6.1 Requirement Changes.....	27
2.6.2 Design Decision Changes.....	28
2.7 Simulink Development History .....	28
<b>3. The Device Controller-Monitor (DCM).....</b>	<b>30</b>
3.1 The DCM.....	30
3.2 Requirements.....	30
3.2.1 Feature Requirements .....	30
3.2.2 Parameter and Mode Input Requirements .....	31
3.3 Design Decisions .....	32
3.3.1 GUI Language Selection .....	32
3.3.2 Welcome Page UI.....	33
3.3.3 Parameter Interface .....	37
3.3.4 Heartview Connection Interface.....	39
3.4 Module Overview.....	41
3.4.1 Purpose of the Module.....	41
3.4.2 List of Public Functions .....	41
3.4.3 Black Box Behavior.....	43
3.4.4 Global Variables .....	44
3.4.5 Private Functions .....	45

3.4.6 Internal Behavior .....	45
3.5 Testing and Results .....	47
3.6 Potential Changes and Future Considerations .....	49
3.6.1 Requirements Changes .....	49
3.6.2 Design Changes .....	50
3.7 DCM Development History .....	52
4. Conclusion .....	53

## List of Figures

Figure 1: Pacemaker Board .....	5
Figure 2: Main Simulink Model for Pacemaker .....	9
Figure 3: Programmable Parameter Subsystem .....	10
Figure 4: Sensing Subsystem .....	11
Figure 5: Microcontroller Pin Mapping Subsystem .....	12
Figure 6: Main Stateflow Chart .....	13
Figure 7: Nested VOO mode Stateflow .....	14
Figure 8: Nested AOO Stateflow .....	15
Figure 9: Nested VVI Stateflow .....	16
Figure 10: Nested AAI Stateflow .....	17
Figure 11: AOO Test 1 .....	19
Figure 12: AOO Test 2 .....	20
Figure 13: VOO Test 1 .....	21
Figure 14: VOO Test 2 .....	22
Figure 15: VVI Test 1 .....	23
Figure 16: VVI Test 2 .....	23
Figure 17: VVI Test 3 .....	24
Figure 18: AAI Test 1 .....	25
Figure 19: AAI Test 2 .....	26
Figure 20: AAI Test 3 .....	26
Figure 21: Python Libraries .....	32
Figure 22: Welcome Page .....	34
Figure 23: Registration Confirmation .....	34
Figure 24 : Welcome confirmation .....	35
Figure 25 : Login/Register functions .....	35
Figure 26: User Data Json File .....	36
Figure 27: Password Encryption Functions .....	36
Figure 28: User Login/Registration Functions .....	36
Figure 29: Main Interface .....	37
Figure 30: Rate limit verification function .....	38

Figure 31: Information pop-up	38
Figure 32: EGM window	39
Figure 33: Main interface code 1	40
Figure 34: Main interface code 2	40
Figure 35: Main interface code 3	40
Figure 36: List of functions	41

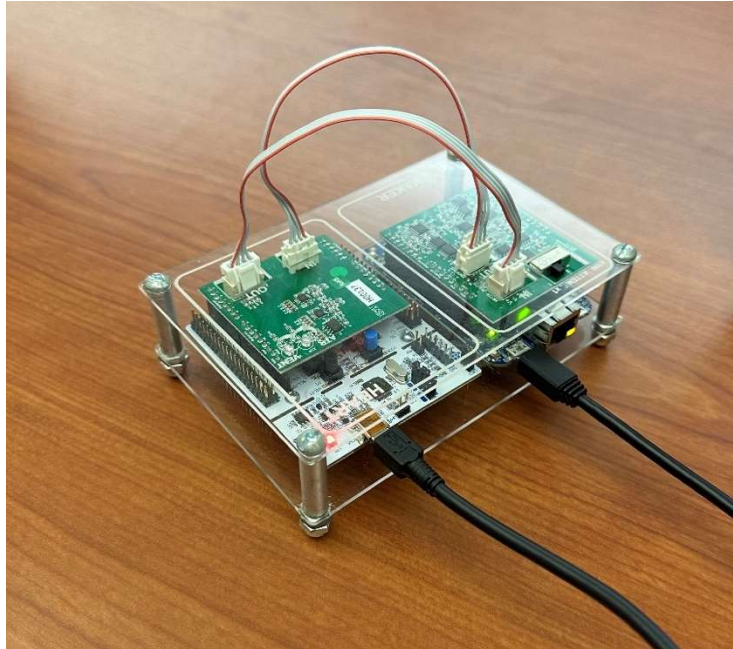
## List of Tables

Table 1: Programmable Parameters for Bradycardia Therapy Modes	6
Table 2: AOO Test 1	19
Table 3: AOO TEST 2	20
Table 4: VOO TEST 1	20
Table 5: VOO TEST 2	21
Table 6: VVI TEST 1	22
Table 7: VVI TEST 2	23
Table 8: VVI TEST 3	24
Table 9: AAI TEST 1	24
Table 10: AAI TEST 2	25
Table 11: AAI TEST 3	26
Table 12: Updated parameters required for each mode	27
Table 13: Simulink Development History	28
Table 14: DCM Testing	48
Table 15: DCM Design Log	52

# 1. Introduction

This document outlines the comprehensive process of designing, developing, testing, and validating a pacemaker system. The project is divided into two main components to ensure a structured approach. The first component focuses on implementing Simulink Stateflows, which are used to model the four operating modes of the pacemaker, representing the backend functionality of the pacemaker. These modes simulate the essential functions that the pacemaker must perform, allowing for control and accurate modelling of heart responses. The second involves developing a Device Controller Monitor (DCM), which includes a graphical user interface (GUI), representing the front-end portion of the pacemaker interface. This interface enables users to interact with the pacemaker, adjusting mode and given parameters as needed.

The primary objective of the project is to create a reliable and safe system that meets the critical requirements of a pacemaker operating in a life-critical environment. This detailed documentation is provided to ensure that every stage of the process is well-documented. This allows for traceability and simplifies the process of making future improvements, or modifications to the system. By following using structured approach, the system is designed to meet both current needs and easily adapt for the future.



*Figure 1: Pacemaker Board*

## 2. Pacemaker

### 2.1 The Pacemaker

The pacemaker is a device designed to regulate abnormal heartbeats by delivering electrical pulses to the heart at a normal rate. The AOO and VOO modes are designed to deliver fixed rate pacing to the heart without monitoring for natural heartbeats, ensuring constant pacing. In AAI and VVI modes, the pacemaker will sense atrial or ventricular activity in the heart and will only deliver electrical pulses when no natural beat is detected. These are the four modes required to be implemented in assignment 1.

### 2.2 Requirements

The following requirements govern the development of the pacemaker operating modes in Simulink:

- **Modes:** The system must support AOO, VOO, AAI, and VVI modes.
- **Programmable Parameters:** The system should allow setting pulse characteristics (pulse width and amplitude) and rate characteristics (lower and upper rate limits, delays).
- **Chamber Pacing:** The design must specify whether the atrium, ventricle, or both chambers are being paced, with appropriate sensing for inhibited modes (AAI and VVI).
- **Hardware Abstraction:** Use a Simulink subsystem to abstract the hardware (pins and interfaces) from the logic, allowing easy modification of pin mapping without affecting the core model.

In this assignment, only four therapy modes are used, AOO, VOO, AAI, and VVI, where a limited number of parameters are used. Shown in Table 1 are the parameters used for these modes.

*Table 1: Programmable Parameters for Bradycardia Therapy Modes*

Parameter	AOO	VOO	AAI	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
VRP				X
ARP			X	
PVARP			X	
Hysteresis			X	X
Rate Smoothing			X	X

## 2.3 Design Decisions

**Stateflow Charts:** Each pacemaker mode (AOO, VOO, AAI, VVI) is designed as a dedicated subchart within the main Simulink Stateflow chart to provide modular and independent control over the operation modes. By creating distinct subcharts for each mode, the design enables clear separation of functionality, making it easier to test, modify, and expand specific modes as required. Each state transition is controlled by well-defined input signals, such as sensing events or rate limits, ensuring that each mode operates based on its specific parameters without interference from other modes.

**Pulse Characteristics:** Adjustable pulse parameters, including pulse width and amplitude, are integrated as input variables to the Stateflow chart, allowing the pacemaker's output characteristics to be configured dynamically. This approach makes it possible to customize pacing settings for different scenarios or patient requirements directly through the input parameters.

**Hardware Hiding:** A separate subsystem was introduced for pin mapping to facilitate hardware abstraction. By isolating the hardware-specific components, such as input and output pin mappings, in a dedicated subsystem, the design enables adaptability across different hardware configurations without the need to alter the primary state chart. This approach to hardware hiding simplifies future updates to the hardware setup, allowing the underlying logic of the pacemaker modes to remain unchanged even if new hardware components are introduced.

**Safety Considerations:** Clear and detailed annotations are included throughout the model to ensure traceability and understanding of the design. Every key component and transition is labeled, and descriptions are provided to clarify functionality and operation. This organized approach prioritizes safety by making it easier to review and verify that each subsystem and operation aligns with safety standards. Additionally, detailed labeling and documentation enhance the maintainability of the model, making it easier for future users or engineers to interpret and validate each element's purpose and functionality.



## 2.4 Simulink Model Overview

### 2.4.1 Main Simulink Model

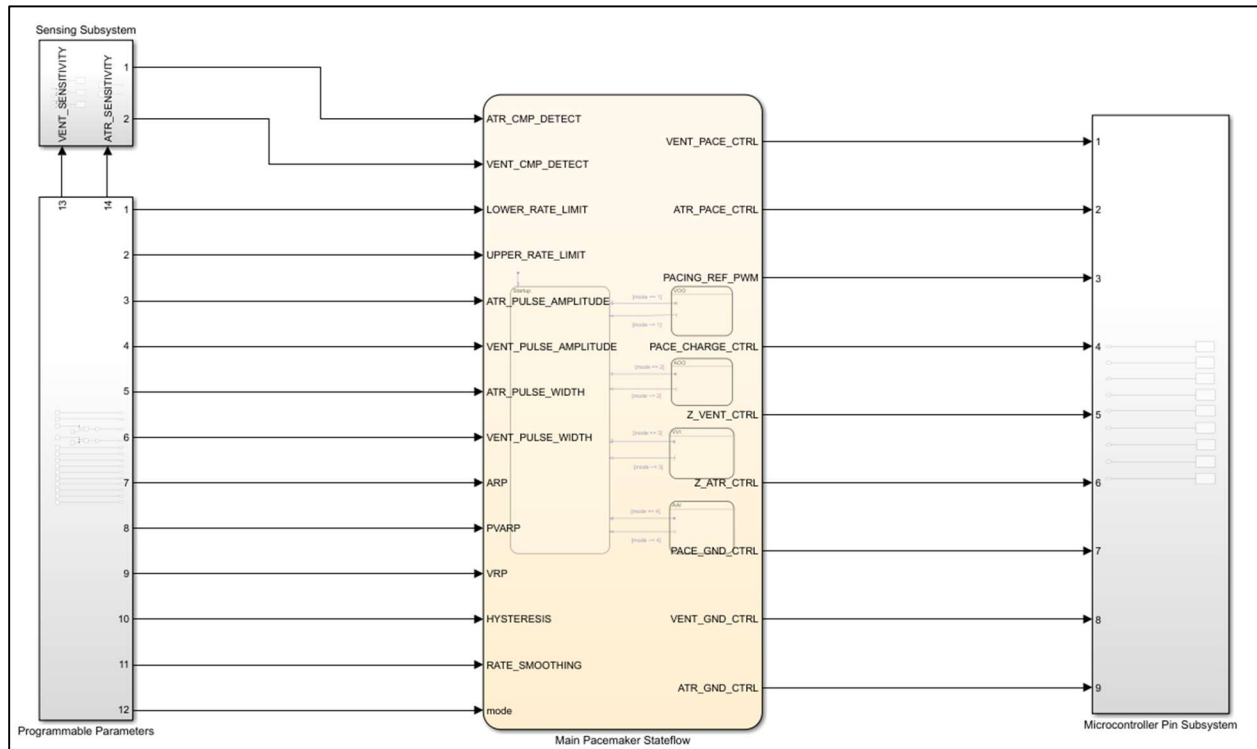


Figure 2: Main Simulink Model for Pacemaker

The main Simulink model is composed of three subsystems and one main Stateflow chart. Each subsystem handles different functional aspects of the pacemaker, including sensing, programmable parameters, and hardware pin mapping. The Stateflow chart contains the core logic of the pacemaker, governing the operational modes (AOO, VOO, AAI, VVI) and transitions between them based on inputs from the subsystems. This modular design allows for flexibility and ease of updating specific components without impacting the entire model.

## 2.4.2 Programmable Parameters Subsystem

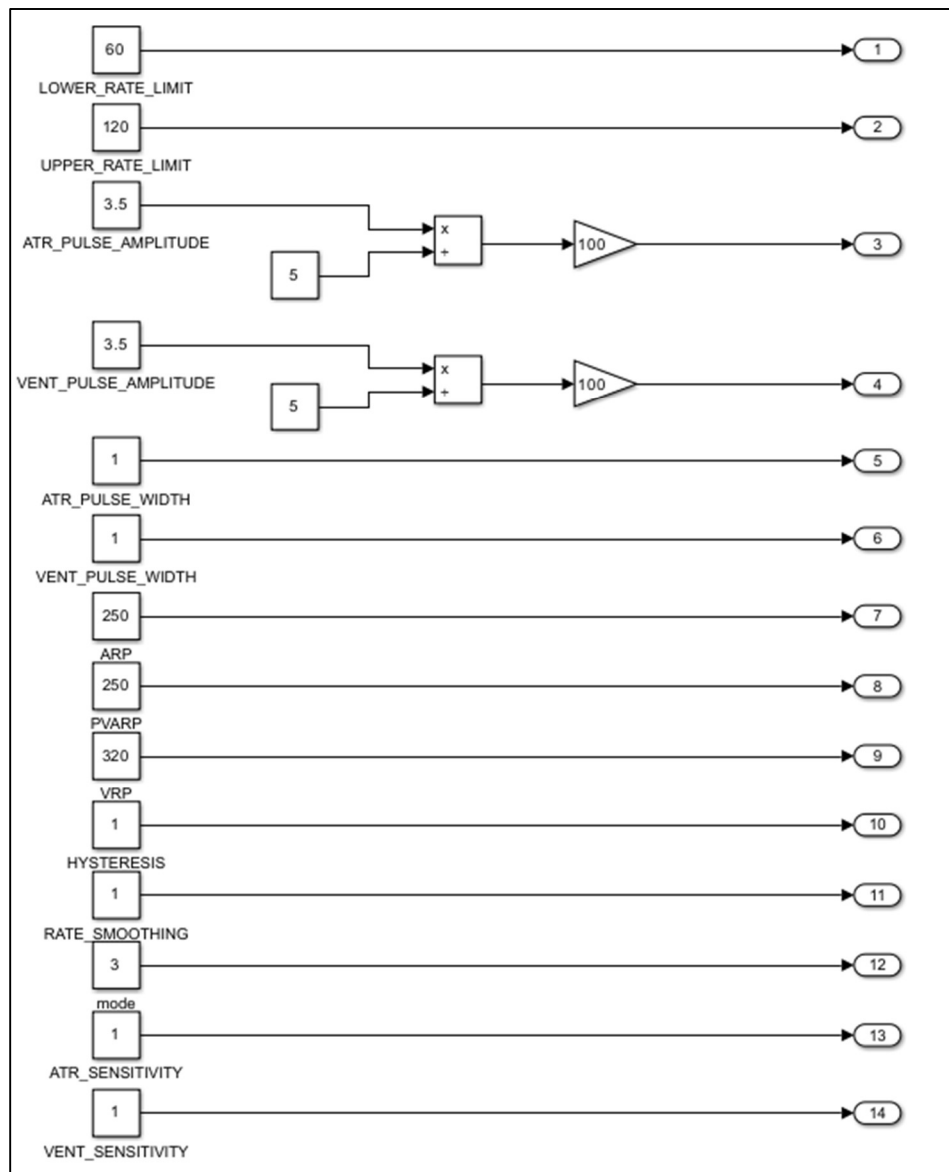


Figure 3: Programmable Parameter Subsystem

The subsystem shown above manages the programmable parameters for the pacemaker. It allows the user to set values related to the four primary modes of operation—AOO, VOO, AAI, and VVI. Key parameters include the lower rate limit, upper rate limit, atrium and ventricle pulse widths, pulse amplitudes, sensitivity levels, and refractory periods (ARP and VRP). Additionally, the subsystem includes inputs for mode selection, as well as parameters designed for future operational modes, such as Boolean inputs for hysteresis and rate smoothing, along with a value for the PVARP.

### 2.4.3 Sensing Subsystem

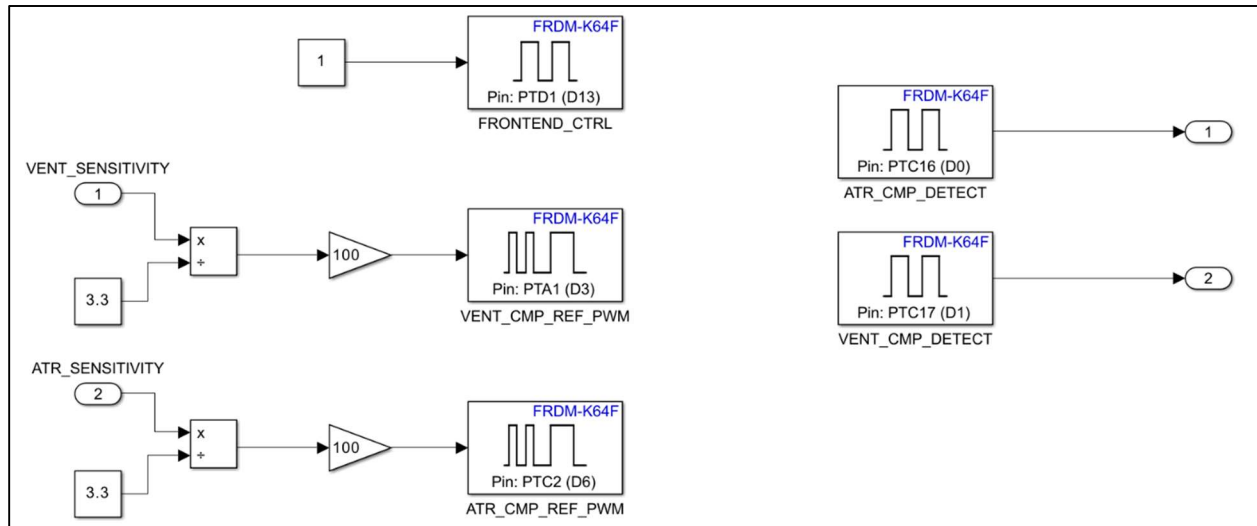


Figure 4: Sensing Subsystem

The diagram above shows the sensing subsystem for the pacemaker. This subsystem processes the inputs VENT\_SENSITIVITY and ATR\_SENSITIVITY by dividing each by the maximum capacitor voltage (3.3V), then applying a gain of 100 to set the duty cycle used to charge the comparison voltage. These processed values are then directed to the appropriate pins for further use in the sensing circuit. The output from this subsystem are VENT\_CMP\_DETECT and ATR\_CMP\_DETECT, that output when activity is sensed from the corresponding lead.

The decision to isolate the sensing subsystem from the main stateflow chart was motivated by a hardware abstraction approach. Since both the PWM generation values and input pins are hardware-dependent, isolating them allows for easier modifications if the hardware changes. By encapsulating this functionality in a separate subsystem, any changes in hardware would require adjustments only within this subsystem, without affecting the main logic in the stateflow. This design enhances modularity and simplifies future hardware adaptations.

## 2.4.4 Microcontroller Pin Mapping Subsystem

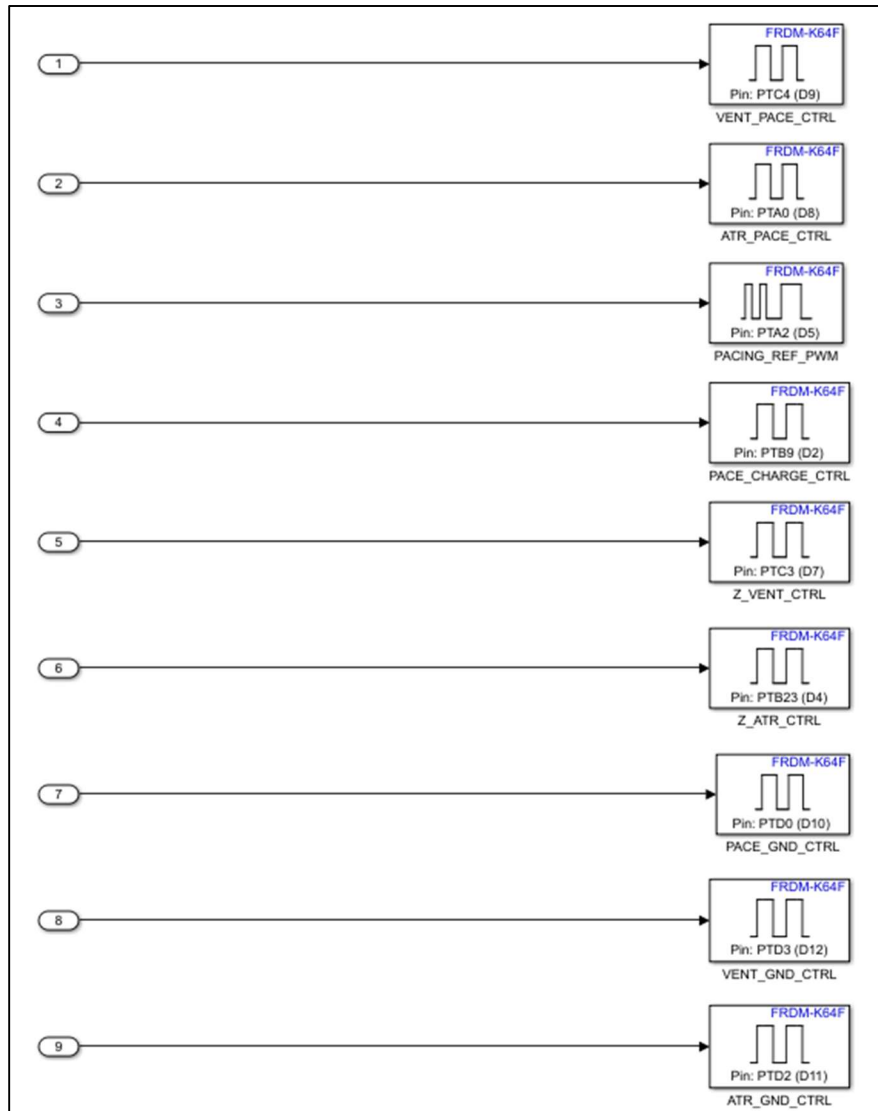


Figure 5: Microcontroller Pin Mapping Subsystem

The subsystem above maps the outputs from the main pacemaker stateflow to the appropriate pins in the microcontroller. It is implemented in a separate subsystem to employ hardware hiding, making it so that this design isolates hardware-specific elements from the core logic of the pacemaker. By having the pin mapping and hardware-dependent components in their own subsystem, the main stateflow logic remains independent of the underlying hardware. This approach enhances modularity and makes the system more adaptable to future hardware changes, as only this subsystem would need to be updated rather than modifying the entire stateflow.

## 2.4.5 Main Stateflow Chart

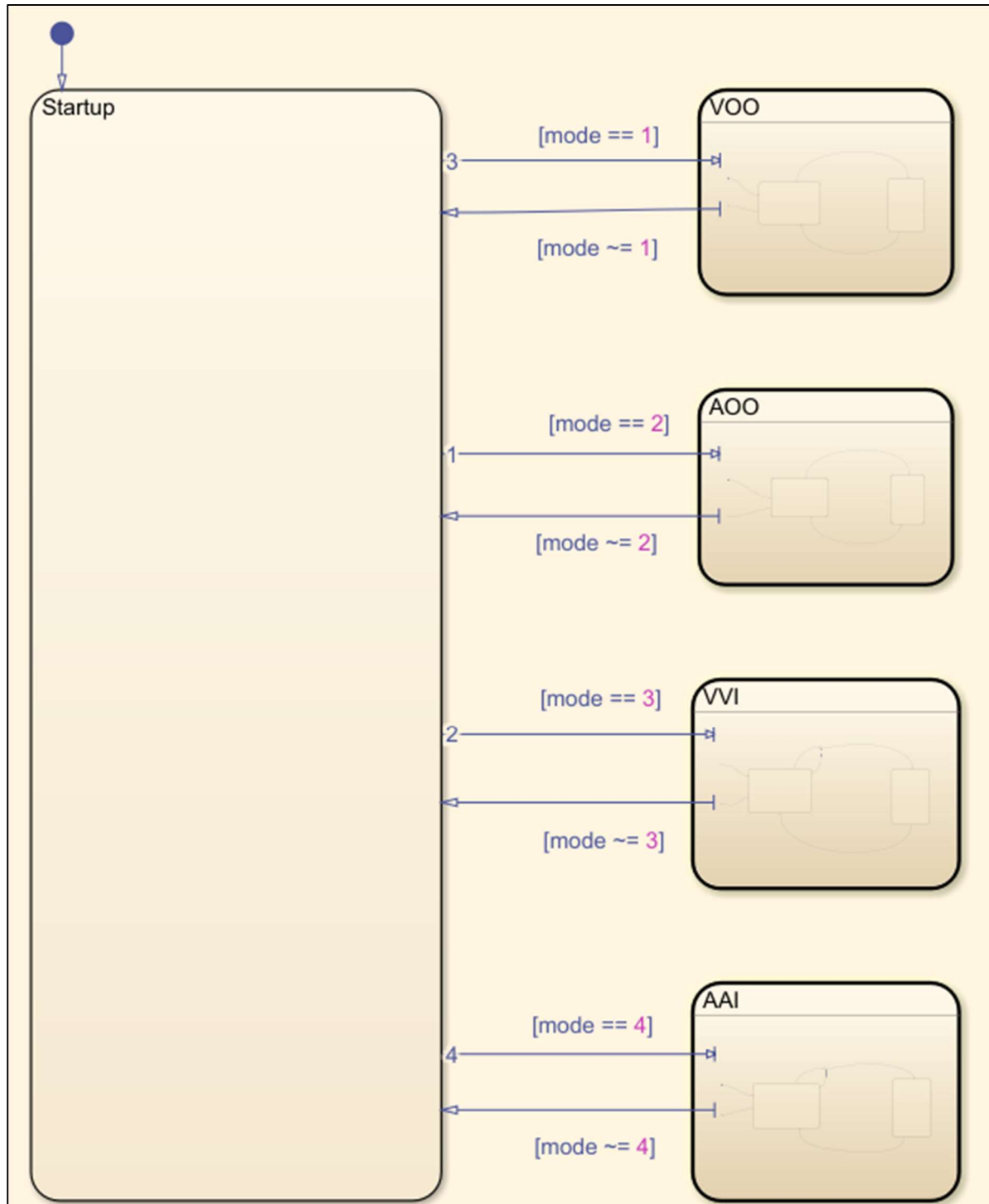


Figure 6: Main Stateflow Chart

The primary Stateflow chart establishes the bradycardia mode for pacemaker operation, with the mode set as an input in the Input Parameter System (Figure 3). Initially, the chart enters a

“Startup” state before transitioning to the bradycardia state specified by one of four modes (modes 1-4). Currently, only a single mode can be active per build, with the pacemaker remaining in that mode until reconfiguration. Future enhancements will support dynamic mode switching, allowing users to select a mode via a switch input, which will prompt a seamless state transition within the chart. The “Startup” mode can also be altered as future features are implemented.

## 2.4.6 VOO Stateflow

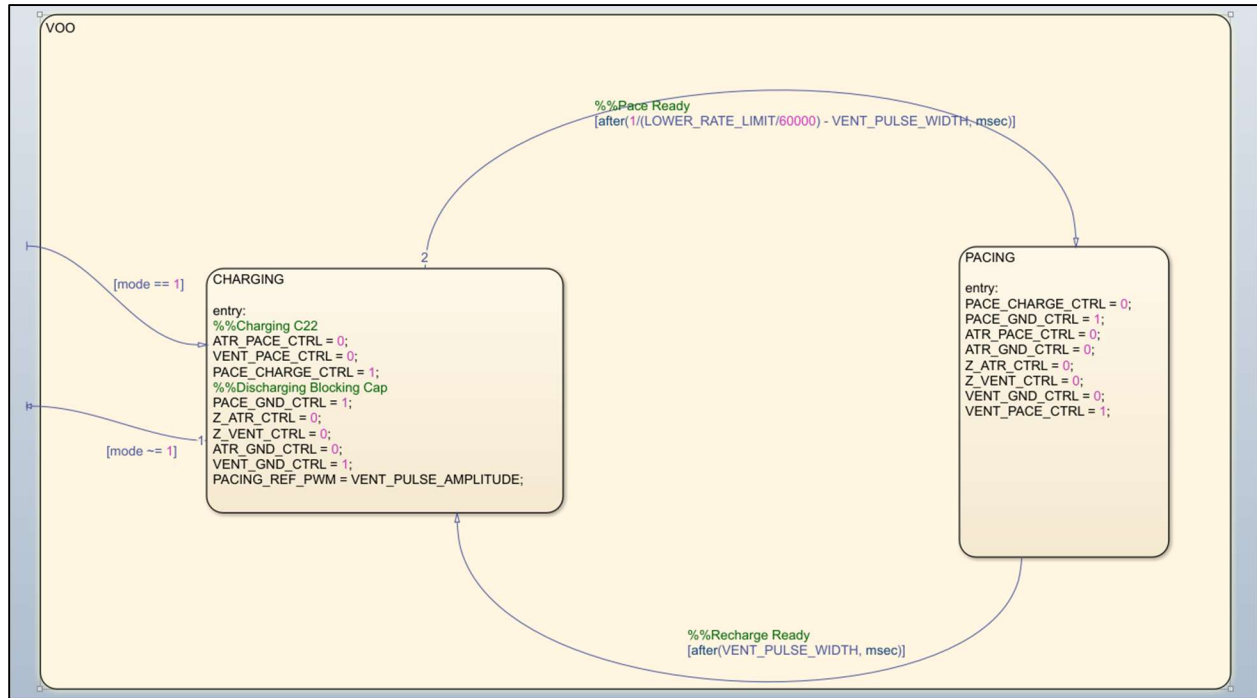


Figure 7: Nested VOO mode Stateflow

**VOO Mode Overview:** The VOO mode provides constant, asynchronous pacing to the ventricle at a rate determined by the lower rate limit, regardless of the natural heartbeat. The pacemaker delivers pulses at this rate, ensuring consistent stimulation without any heartbeat sensing.

**Charging/Discharging State:** The charging/discharging state controls the charging of capacitor C22, which stores the pacing energy, and the discharging of the blocking capacitor C1. When PACE\_CHARGE\_CTRL is set to high, current flows into C22. The voltage stored in C22 after charging is determined by the user-defined VENT\_PULSE\_AMPLITUDE, which is divided by 5 and scaled by 100 before being applied to PACING\_REF\_PWM to set the charge level. To discharge C21, VENT\_GND\_CTRL and PACE\_GND\_CTRL are set to high, allowing the capacitor to discharge.

**Pacing State:** The pacing state controls the delivery of the pacing pulse to the ventricle. When VENT\_PACE\_CTRL is set high, capacitor C22 discharges into the ventricle, delivering a pacing

pulse. This also charges the blocking capacitor C21, ensuring that current flows only in the intended direction.

**Stateflow Timing:** The state transitions from charging to pulsing based on the lower rate limit, which is input in pulses per minute and divided by 60000 to yield pulses per millisecond. Dividing 1 by this value gives the total period. The state switches from charging to pulsing after a duration equal to the period minus the pulse width, then back to charging after the pulse width time. Upon each state entry, the output controls adjust according to the operation requirements.

## 2.4.7 AOO Stateflow

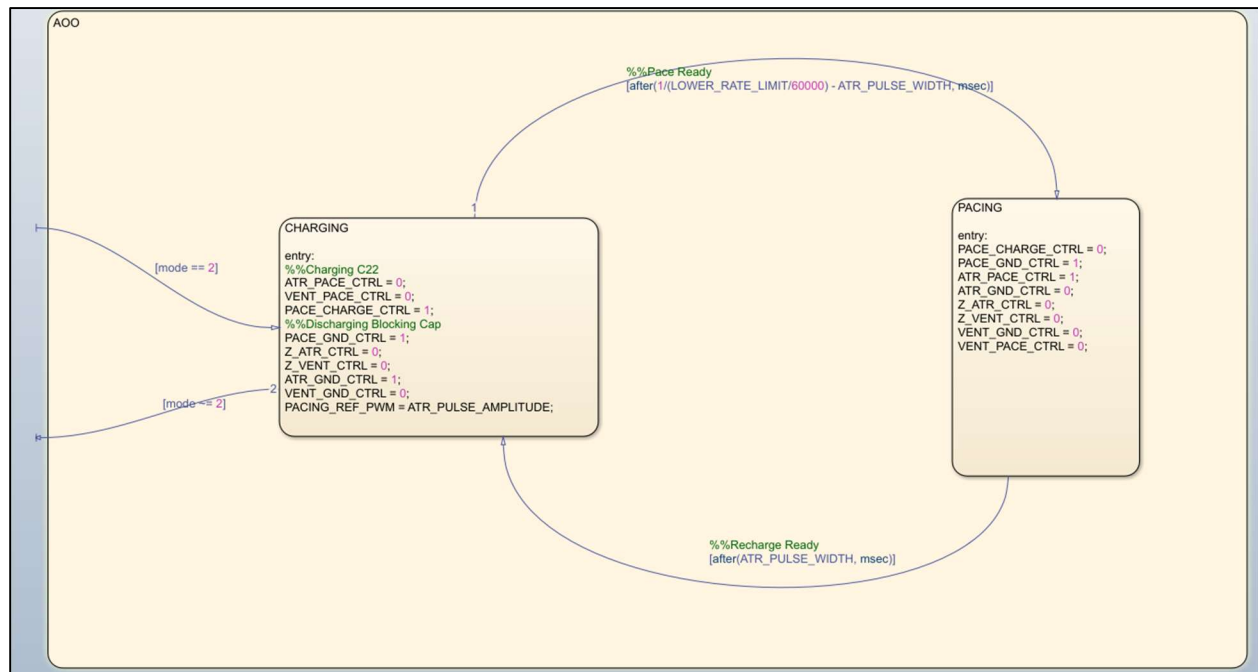


Figure 8: Nested AOO Stateflow

**AOO Mode Overview:** The AOO mode provides constant, asynchronous pacing to the atrium at a rate determined by the lower rate limit, regardless of the natural heartbeat. The pacemaker delivers pulses at this rate, ensuring consistent stimulation without any heartbeat sensing.

**Charging/Discharging State:** The charging/discharging state controls the charging of capacitor C22, which stores the pacing energy, and the discharging of the blocking capacitor C1. When PACE\_CHARGE\_CTRL is set to high, current flows into C22. The voltage stored in C22 after charging is determined by the user-defined ATR\_PULSE\_AMPLITUDE, which is divided by 5 and scaled by 100 before being applied to PACING\_REF\_PWM to set the charge level. To discharge C21, VENT\_GND\_CTRL and PACE\_GND\_CTRL are set to high, allowing the capacitor to discharge.

**Pacing State:** The pacing state controls the delivery of the pacing pulse to the atrium. When ATR\_PACE\_CTRL is set high, capacitor C22 discharges into the atrium, delivering a pacing pulse. This also charges the blocking capacitor C21, ensuring that current flows only in the intended direction.

**Stateflow Timing:** The state transitions from charging to pulsing based on the lower rate limit, which is input in pulses per minute and divided by 60000 to yield pulses per millisecond. Dividing 1 by this value gives the total period. The state switches from charging to pulsing after a duration equal to the period minus the pulse width, then back to charging after the pulse width time. Upon each state entry, the output controls adjust according to the operation requirements.

## 2.4.8 VVI Stateflow

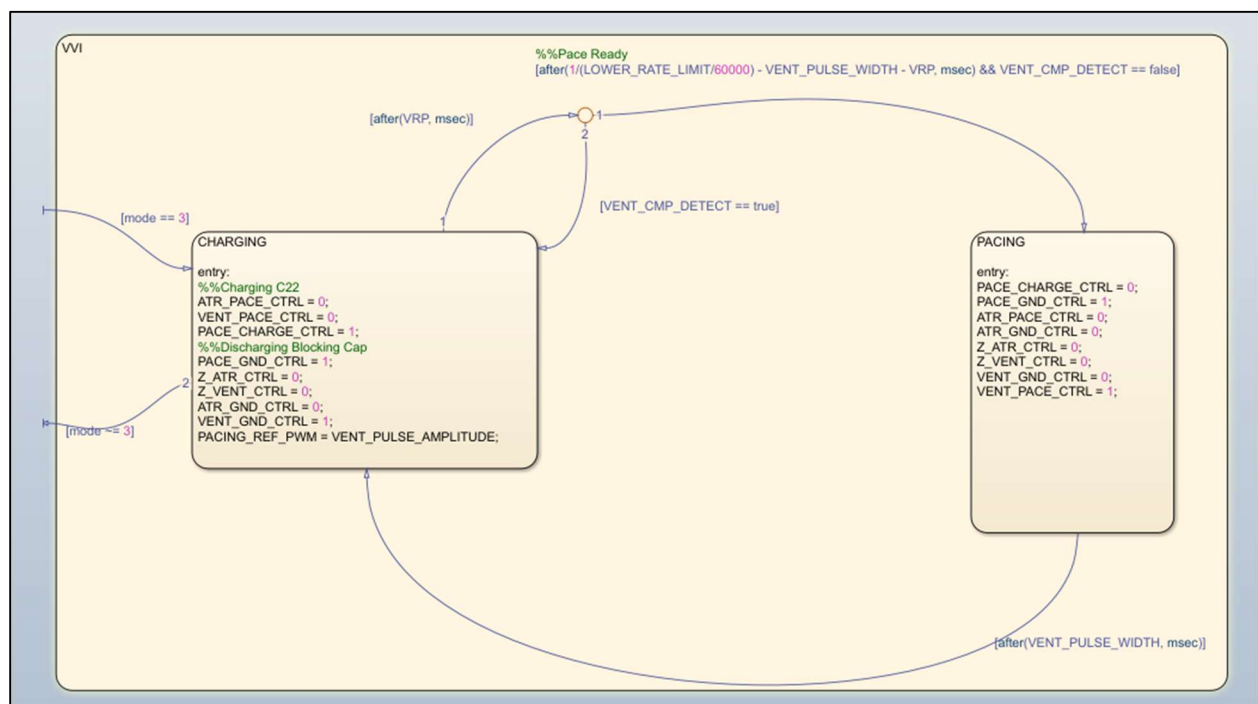


Figure 9: Nested VVI Stateflow

**VVI Mode Overview:** The VVI mode provides ventricular pacing at a rate determined by the lower rate limit, with the added capability to sense natural ventricular activity and inhibit pacing when appropriate. This mode outputs pulses at the specified rate only when no intrinsic heartbeat is detected, ensuring pacing support when needed and preventing unnecessary stimulation.

**Sensing:** In VVI mode, ventricular sensing is employed to determine the appropriate timing for pacing. This process relies on the output from the VENT\_CMP\_DETECT pin within the sensing subsystem. The threshold voltage for comparison is established based on the input from VENT\_SENSITIVITY, which is scaled to generate a PWM signal at VENT\_CMP\_REF\_PWM to charge the comparison capacitor.



**Pacing State:** The pacing state controls the delivery of the pacing pulse to the ventricle. When VENT\_PACE\_CTRL is set high, capacitor C22 discharges into the ventricle, delivering a pacing pulse. This also charges the blocking capacitor C21, ensuring that current flows only in the intended direction.

### 2.4.9 AAI Stateflow



**AAI Mode Overview:** The AAI mode provides atrial pacing at a rate determined by the lower rate limit, with the added capability to sense natural atrial activity and inhibit pacing when appropriate. This mode outputs pulses at the specified rate only when no intrinsic heartbeat is detected, ensuring pacing support when needed and preventing unnecessary stimulation.

**Sensing:** In AAI mode, atrial sensing is employed to determine the appropriate timing for pacing. This process relies on the output from the ATR\_CMP\_DETECT pin within the sensing subsystem. The threshold voltage for comparison is established based on the input from ATR\_SENSITIVITY, which is scaled to generate a PWM signal at ATR\_CMP\_REF\_PWM to charge the comparison capacitor.

**Charging/Discharging State:** The charging/discharging state controls the charging of capacitor C22, which stores the pacing energy, and the discharging of the blocking capacitor C1. When PACE\_CHARGE\_CTRL is set to high, current flows into C22. The voltage stored in C22 after charging is determined by the user-defined ATR\_PULSE\_AMPLITUDE, which is divided by 5 and scaled by 100 before being applied to PACING\_REF\_PWM to set the charge level. To discharge C21, VENT\_GND\_CTRL and PACE\_GND\_CTRL are set to high, allowing the capacitor to discharge.

**Pacing State:** The pacing state controls the delivery of the pacing pulse to the atrium. When ATR\_PACE\_CTRL is set high, capacitor C22 discharges into the atrium, delivering a pacing pulse. This also charges the blocking capacitor C21, ensuring that current flows only in the intended direction.

**Stateflow Timing:** The transition between states from charging to pulsing is governed by the lower rate limit, specified in pulses per minute and converted to pulses per millisecond by dividing by 60,000. The inverse of this value determines the total period. The system switches from the charging state to a junction state after a duration defined by the atrial refractory period (ARP). At this junction, if the ATR\_CMP\_DETECT input pin reads high, the state reverts to charging, allowing for the ARP duration to elapse. Conversely, if no intrinsic activity is detected, the state transitions to pulsing after a duration equal to the total period minus the pulse width and ARP, returning to charging following the pulse width interval. This is to ensure the paces are delivered at the specified rate. With each state entry, the output controls are adjusted to align with the operational requirements.

## 2.5 Testing and Results

Testing was conducted using the provided pacemaker board and the HeartView desktop application to observe pacing signals. The following tests were performed:

- **Mode Switching:** Verified correct pacing behavior for each mode (AOO, VOO, AAI, VVI) by observing the pacing signal output.
- **Heart Rate Fluctuation:** Verified correct sensing/pacing behavior for changes in BPM (Under and over the lower rate limit).
- **Parameter Adjustments:** Tested the system's response to changes in pulse width, amplitude, and rate limits.
- **Pin Mapping Validation:** Confirmed that pin mapping abstraction worked correctly, with the correct hardware interface mapped without modifying the core model.

Testing was performed through extensive trial and error with the pacemaker and heart view outputs. By constantly viewing different waveform models and comparing results to PACEMAKER documents, we were able to conclude whether our systems met the required specifications of each mode.

Table 2: AOO Test 1

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: AOO	Natural Atrium: ON
Pulse Width: 1ms	Natural Ventricle: OFF
Pulse Amplitude: 3.5V	Natural Heart Rate: 118 bpm
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker paces the atrium consistently while the simulated heart atrium is active	
Actual Output: There is no AOO pacing showing up at all	
Fix: Wrong pins enabled in the state chart, PACE_CHARGE_CTRL = 0 should be enabled	

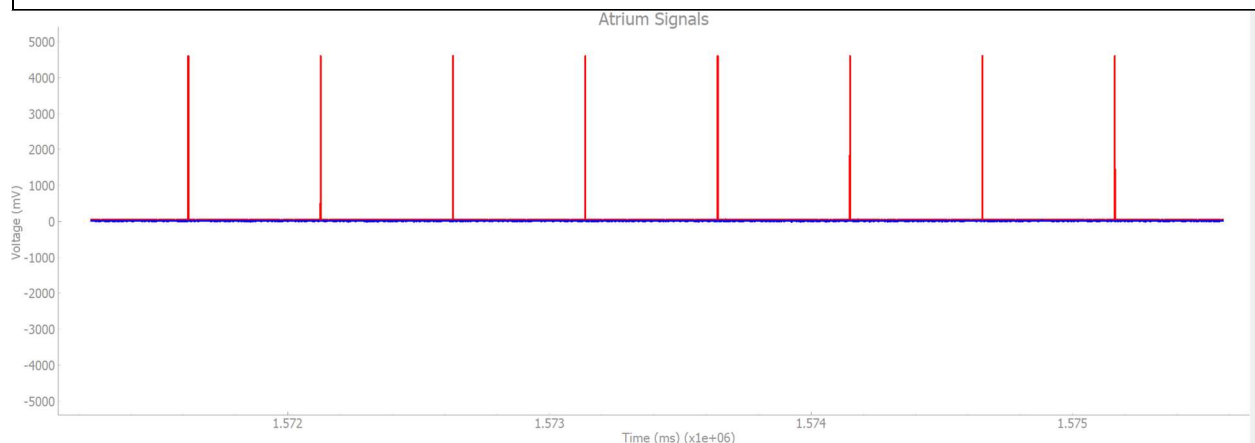


Figure 11: AOO Test 1

Table 3: AOO TEST 2

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: AOO	Natural Atrium: ON
Pulse Width: 1ms	Natural Ventricle: OFF
Pulse Amplitude: 3.5V	Natural Heart Rate: 77 bpm
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker paces the atrium consistently while the simulated heart atrium is active	
Actual Output: The pacemaker paces the atrium consistently while the simulated heart atrium is active	

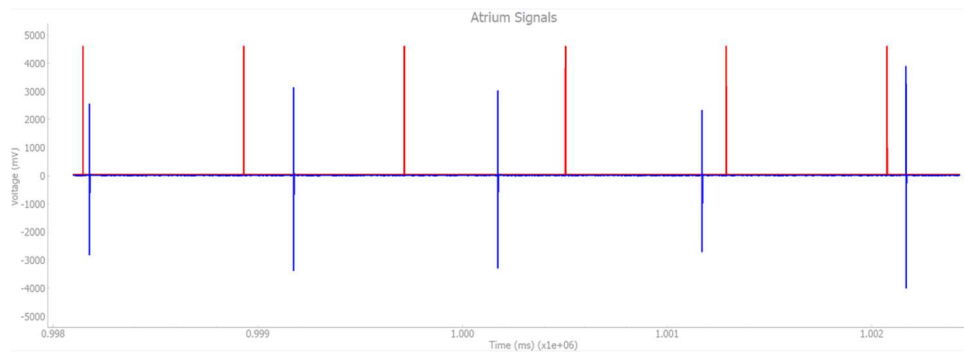


Figure 12: AOO Test 2

Table 4: VOO TEST 1

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: VOO	Natural Atrium: OFF
Pulse Width: 1ms	Natural Ventricle: ON
Pulse Amplitude: 3.5V	Natural Heart Rate: 118 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker paces the ventricle consistently while the simulated heart ventricle is active	

Actual Output: Pacing but timing errors cause constant pacing

Fix: Transition state timing was wrong, didn't include width and lower rate

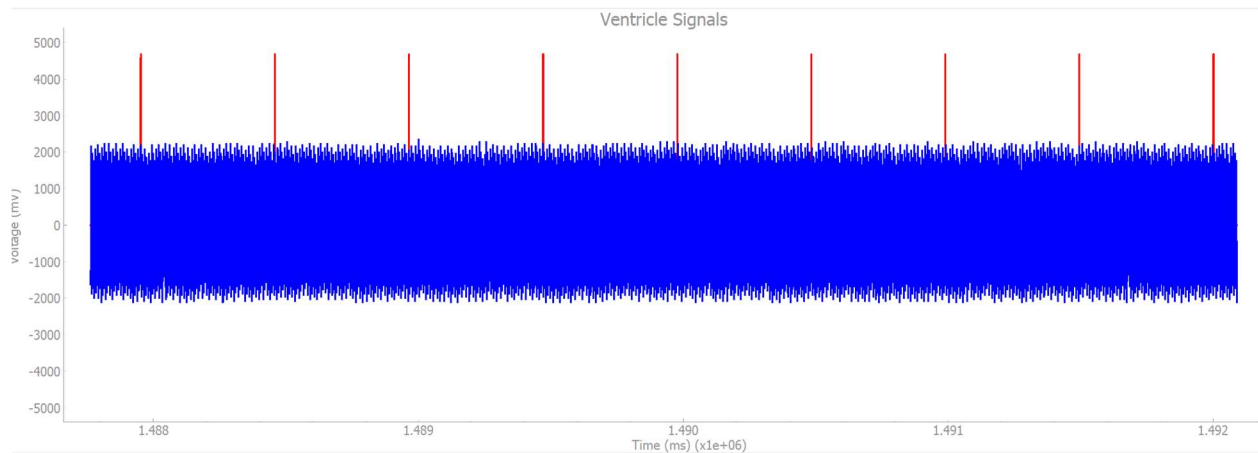


Figure 13: VOO Test 1

Table 5: VOO TEST 2

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: VOO	Natural Atrium: OFF
Pulse Width: 1ms	Natural Ventricle: ON
Pulse Amplitude: 3.5V	Natural Heart Rate: 76 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker paces the ventricle consistently while the simulated heart ventricle is active	
Actual Output: The pacemaker paces the ventricle consistently while the simulated heart ventricle is active	

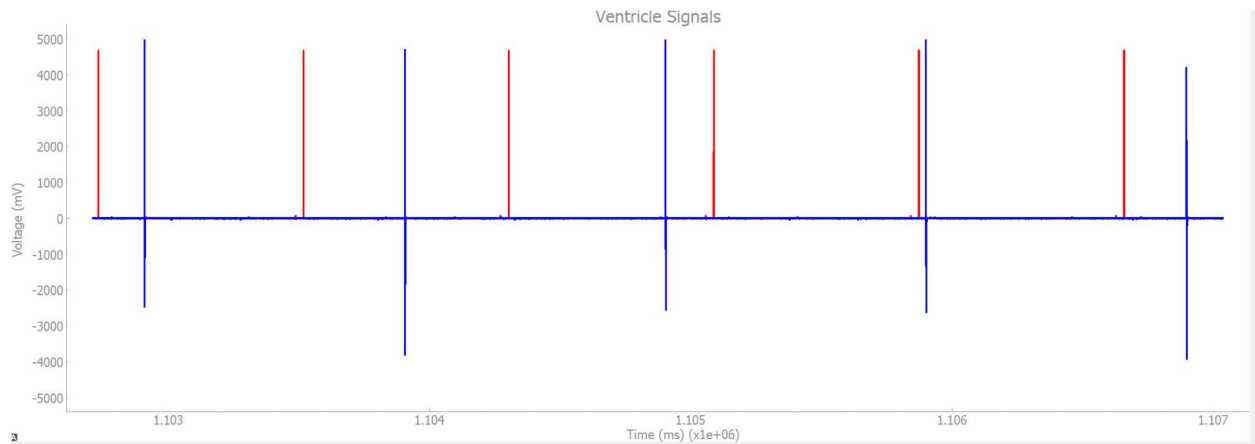


Figure 14: VOO Test 2

Table 6: VVI TEST 1

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: VVI	Natural Atrium: OFF
Pulse Width: 1ms	Natural Ventricle: ON
Pulse Amplitude: 3.5V	Natural Heart Rate: 118 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker senses the simulated heart ventricle beating above 60 bpm and does not pace the ventricle	
Actual Output: Doesn't pace but every couple thousand ms, a pace waveform will show	
Fix: Lower Rate Limit and Sensitivity not implemented correctly	

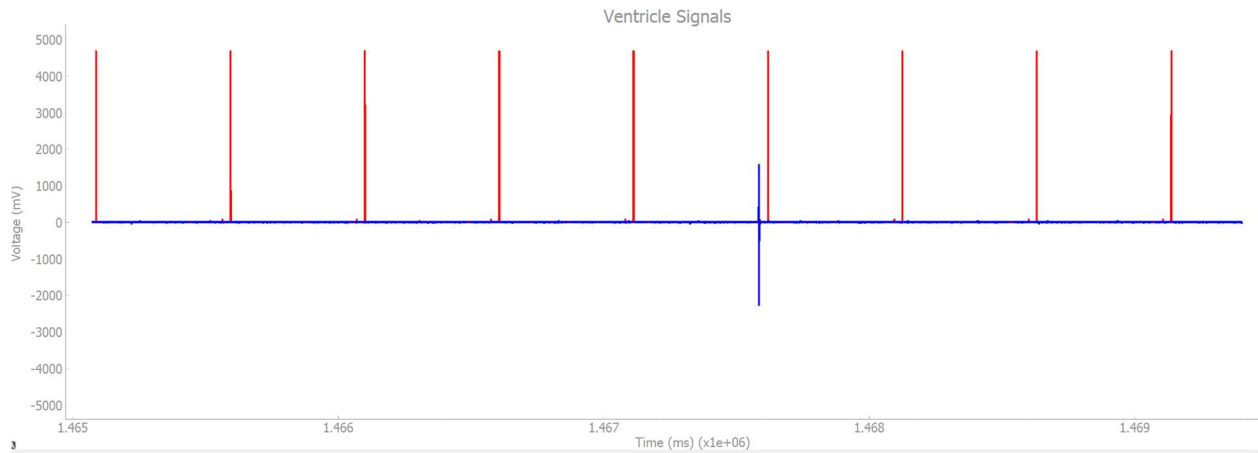


Figure 15: VVI Test 1

Table 7: VVI TEST 2

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: VVI	Natural Atrium: OFF
Pulse Width: 1ms	Natural Ventricle: ON
Pulse Amplitude: 3.5V	Natural Heart Rate: 30 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker paces the ventricle consistently while the simulated heart ventricle is below 60 bpm	
Actual Output: The pacemaker paces the ventricle consistently while the simulated heart ventricle is below 60 bpm	

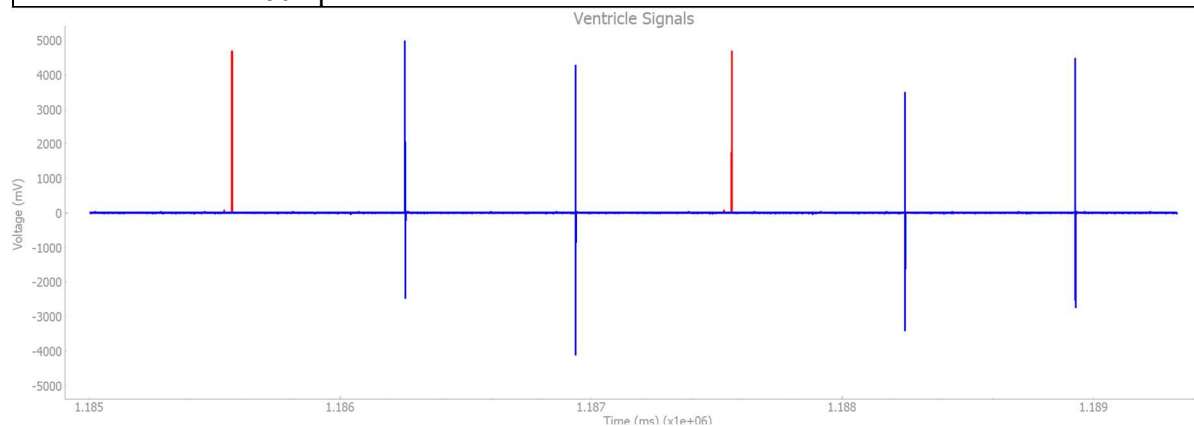


Figure 16: VVI Test 2

Table 8: VVI TEST 3

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: VVI	Natural Atrium: OFF
Pulse Width: 1ms	Natural Ventricle: ON
Pulse Amplitude: 3.5 V	Natural Heart Rate: 130 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker senses the simulated heart ventricle beating above 60 bpm and does not pace the ventricle	
Actual Output: The pacemaker senses the simulated heart ventricle beating above 60 bpm and does not pace the ventricle	

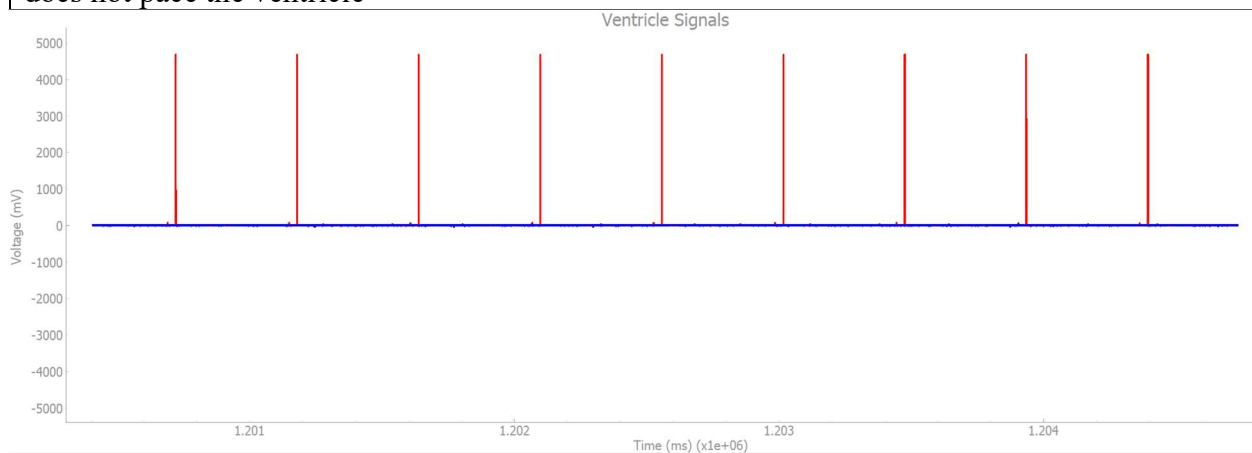


Figure 17: VVI Test 3

Table 9: AAI TEST 1

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: AAI	Natural Atrium: ON
Pulse Width: 1ms	Natural Ventricle: OFF
Pulse Amplitude: 3.5V	Natural Heart Rate: 142 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker senses the simulated heart atrial beating above 60 bpm and does not pace the atrial	
Actual Output: At 142, the pacemaker still paces even though this heart rate is greater than the lower rate	



Fix: Sensitivity and Lower Rate values entered incorrectly, but theory correct

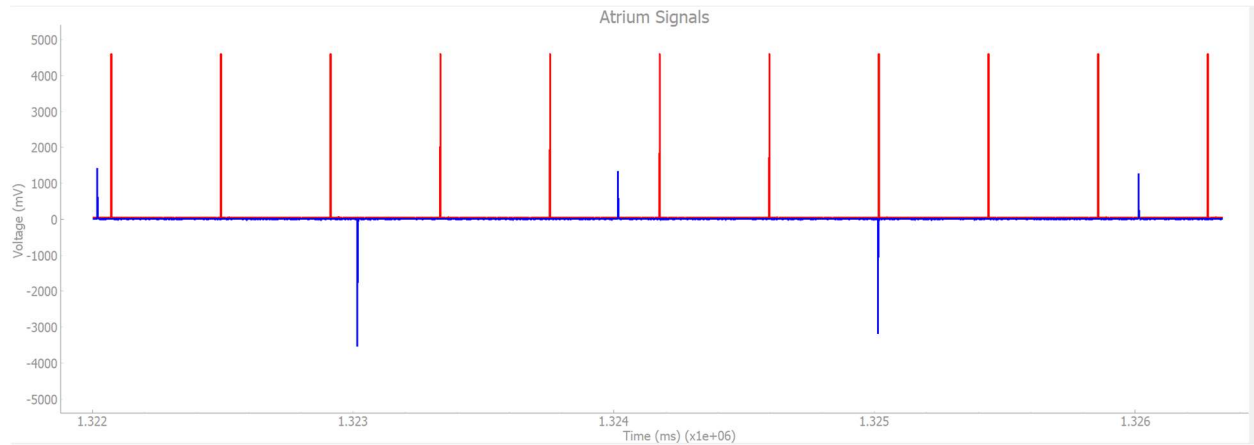


Figure 18: AAI Test 1

Table 10: AAI TEST 2

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: AAI	Natural Atrium: ON
Pulse Width: 1ms	Natural Ventricle: OFF
Pulse Amplitude: 3.5V	Natural Heart Rate: 30 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker paces the atrial consistently while the simulated heart atrial is below 60 bpm	
Actual Output: The pacemaker paces the atrial consistently while the simulated heart atrial is below 60 bpm	

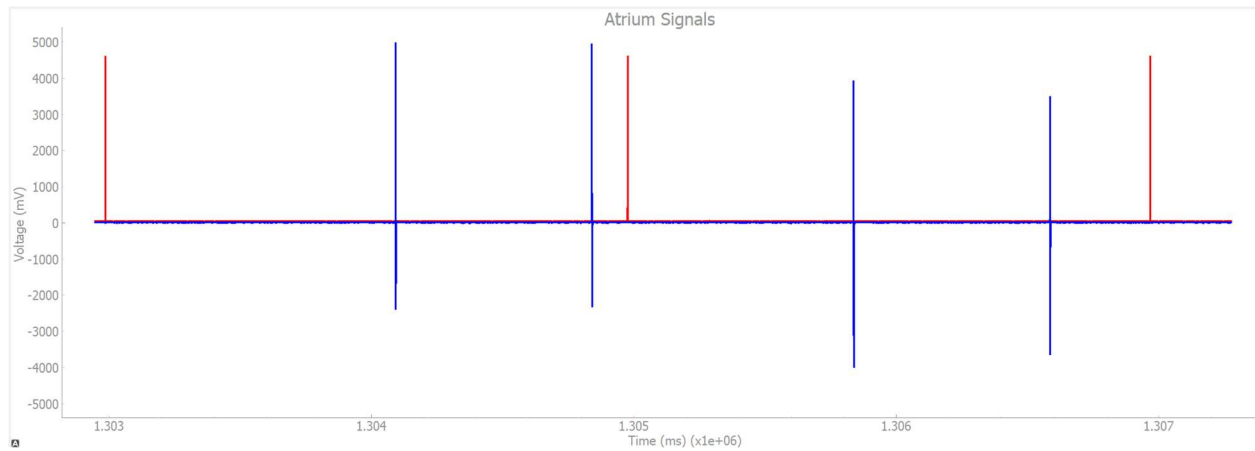


Figure 19: AAI Test 2

Table 11: AAI TEST 3

<u>Pacemaker Settings</u>	<u>Heartview Settings</u>
Mode: AAI	Natural Atrium: ON
Pulse Width: 1ms	Natural Ventricle: OFF
Pulse Amplitude: 3.5V	Natural Heart Rate: 130 BPM
Lower Rate: 60 ppm	Natural AV Delay: 30ms
Expected Output: The pacemaker senses the simulated heart atrial beating above 60 bpm and does not pace the atrial	
Actual Output: The pacemaker senses the simulated heart atrial beating above 60 bpm and does not pace the atrial	

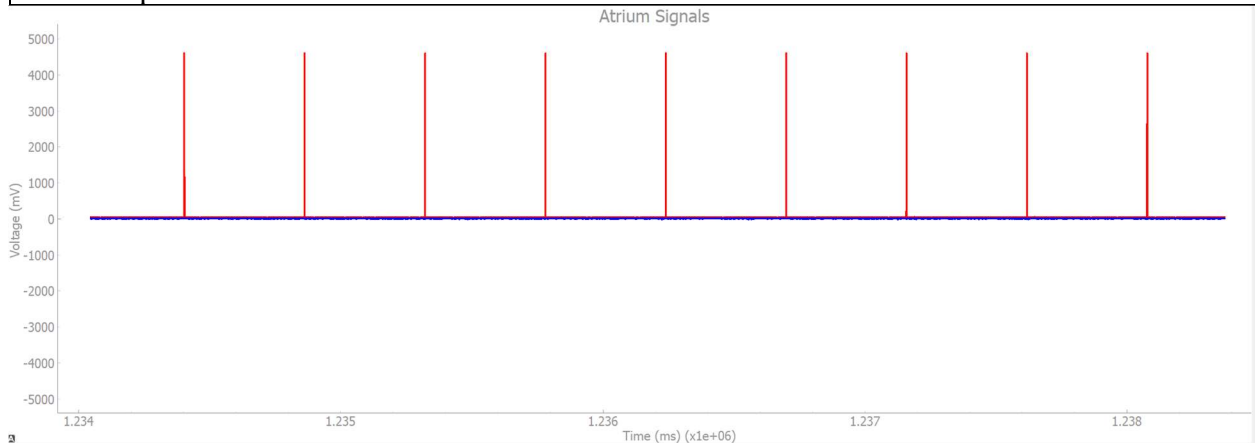


Figure 20: AAI Test 3

## 2.6 Potential Changes and Future Considerations

### 2.6.1 Requirement Changes

- Additional support will be added for AOOR, VOOR, AAIR, VVIR modes.
- Parameters will be added for the additional modes' rate modulation functionality.
- Possible changes to the pacing rate limits or additional parameters may be introduced in future versions.
- The system will have to establish communication with and accept input values from the DCM and apply them to the simulation model

Table 12: Updated parameters required for each mode

Parameter	AOOR	AAIR	VOOR	VVIR
Lower Rate Limit	X	X	X	X
Upper Rate Limit	X	X	X	X
Maximum Sensor Rate	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
VRP				X
ARP		X		
PVARP		X		
Hysteresis		X		X
Rate Smoothing		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

## 2.6.2 Design Decision Changes

- The hardware abstraction model may evolve as more complex hardware is introduced, necessitating updates to the subsystem.
- Introduce additional safety requirements, such as a failsafe mode or error detection for out-of-range inputs, to ensure that pacing parameters do not exceed safe limits.
- May add additional states or sub-states in the state flow to handle safety conditions, possibly including real-time monitoring of signal integrity.
- Add a more intuitive way to transition between modes.
- Expand the stateflow to accommodate for additional modes.

## 2.7 Simulink Development History

Table 13: Simulink Development History

Date	Changes
September 26 <sup>th</sup> , 2024	Created initial stateflows for VOO, AOO modes. Gathered parameters for each mode from documentation.
October 3 <sup>rd</sup> , 2024	Faced issues with VOO and AOO initial stateflows. Switched PACING_REF_PWM to a PWM Output, as it was a Digital Output before which caused undesired functionality.  Tested AOO and VOO, verified correct functionality of each function.
October 4 <sup>th</sup> , 2024	Created initial stateflows for VVI, AAI with incorrect timing and sensing. Gathered parameters for VVI and AAI from documentation.
October 10 <sup>th</sup> , 2024	Created an initial combined model to implement hardware hiding. Included a programmable parameters subsystem which was inputted to the main pacemaker stateflow. Also created a microcontroller pin subsystem that takes the output values programmed from the stateflow as input.  Set FRONTEND_CTRL to HIGH in the programmable parameters subsystem. FRONTEND_CTRL wasn't included previously, resulting in incorrect sensing.
October 16 <sup>th</sup> , 2024	Corrected timing for VVI, AAI by utilizing the VRP/ARP in the transition to pacing.  Confirmed correctness of VVI and AAI models.

October 22 <sup>nd</sup> , 2024	<p>Removed sensing parameters from programmable parameters subsystem, added them to a separate sensing subsystem.</p> <p>Added AOO, VOO, AAI, VVI stateflows to the combined model. Confirmed correctness of each mode in the combined model by switching the mode value.</p>
---------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 3. The Device Controller-Monitor (DCM)

### 3.1 The DCM

The Device Controller-Monitor (DCM) is a user application used to manage and interact with the pacemaker model operating on an FRDM-K64F microcontroller. Through a graphical user interface (GUI), the DCM provides users with an easy way to adjust the pacemaker's programmable settings and select the modes available to be run on the Pacemaker. This interface includes interactive elements such as sliders for adjusting parameter values, drop-downs for mode selections, and other accessibility features to enhance user experience. In assignment 2, the DCM's Communication Layer will translate user inputs to the pacemaker, establishing a reliable two-way link with the microcontroller. This setup allows for simple control of the pacemaker's parameters and modes from a user-friendly desktop application.

### 3.2 Requirements

This section outlines the requirements for developing the Device Controller-Monitor (DCM) interface. The DCM should be user-friendly, contain user security features and provide the necessary inputs for managing pacemaker parameters and modes. The DCM must contain the following elements ensure these requirements are met.

#### 3.2.1 Feature Requirements

1. Welcome Screen:
  - The welcome screen must enable the user with the ability to log in, (if the user has previously registered on the platform) or register if they are a new user.
  - The Welcome screen must allow a maximum of 10 users to register.
2. User interface:
  - The interface shall support managing windows display both text and graphical content.

- The interface must allow the user to interact using input buttons and process their positioning.
  - The interface will display all programmable parameters for easy review and modification.
  - The DCM will include a visual indicator to show when communication between the DCM and the pacemaker device is active.
3. Device communication status:
- The DCM must indicate visually when the DCM is actively communicating with the pacemaker device.

### **3.2.2 Parameter and Mode Input Requirements**

1. The DCM must provide the user with an option to select one of the following pacemaker modes:
  - VOO
  - AOO
  - VVI
  - AAI
2. The DCM must contain programmable pacemaker parameters and allow users to modify values of the following:
  - Lower Rate Limit
  - Upper Rate Limit
  - Atrial Amplitude
  - Atrial Pulse Width
  - Ventricular Amplitude
  - Ventricular Pulse Width
  - VRP (Ventricular Refractory Period)

- ARP (Atrial Refractory Period)
3. The DCM must make provision for storing all the parameters listed above including the mode, allowing for future use of these variables.

## 3.3 Design Decisions

### 3.3.1 GUI Language Selection

A graphical user interface (GUI) was chosen as the method for constructing the DCM to provide an intuitive and interactive way for users to manage the pacemaker settings.

Python was selected as the programming language due to its simplicity and the availability of built-in libraries, such as Tkinter, which would assist in streamlining the development process of the graphical user interface. Python's simplicity and ease of use make it an ideal choice for accomplishing the principal goal of creating a user-friendly environment, while still being powerful enough to meet the project's functional requirements.

```
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
import json
import tkinter.font as tkFont
import bcrypt
```

Figure 21: Python Libraries

Figure 10 provides the code used to import all the Python libraries that were used to create the DCM for assignment one. These libraries were chosen for the GUI's construction because they each offer unique features that contribute to a user-friendly and secure interface. By combining their individual capabilities, we're able to efficiently build a responsive and visually appealing GUI with secure data management.



The **Tkinter** library, Python's standard library for creating GUIs, allows for rapid development of graphical elements, making it well-suited for building interactive user interfaces. Tkinter's **messagebox** module enables the easy creation of pop-up alerts and dialogues, which are helpful for improving the user experience by providing feedback and notifications. This was used multiple times in the assignment when notifying the user when their sign-in request was successful, when their registration was successful when their parameters were saved and others. Additionally, **ttk**, which is a module within tkinter, supports advanced styling for elements and widgets, which gives the interface a more modern and polished look and feel.

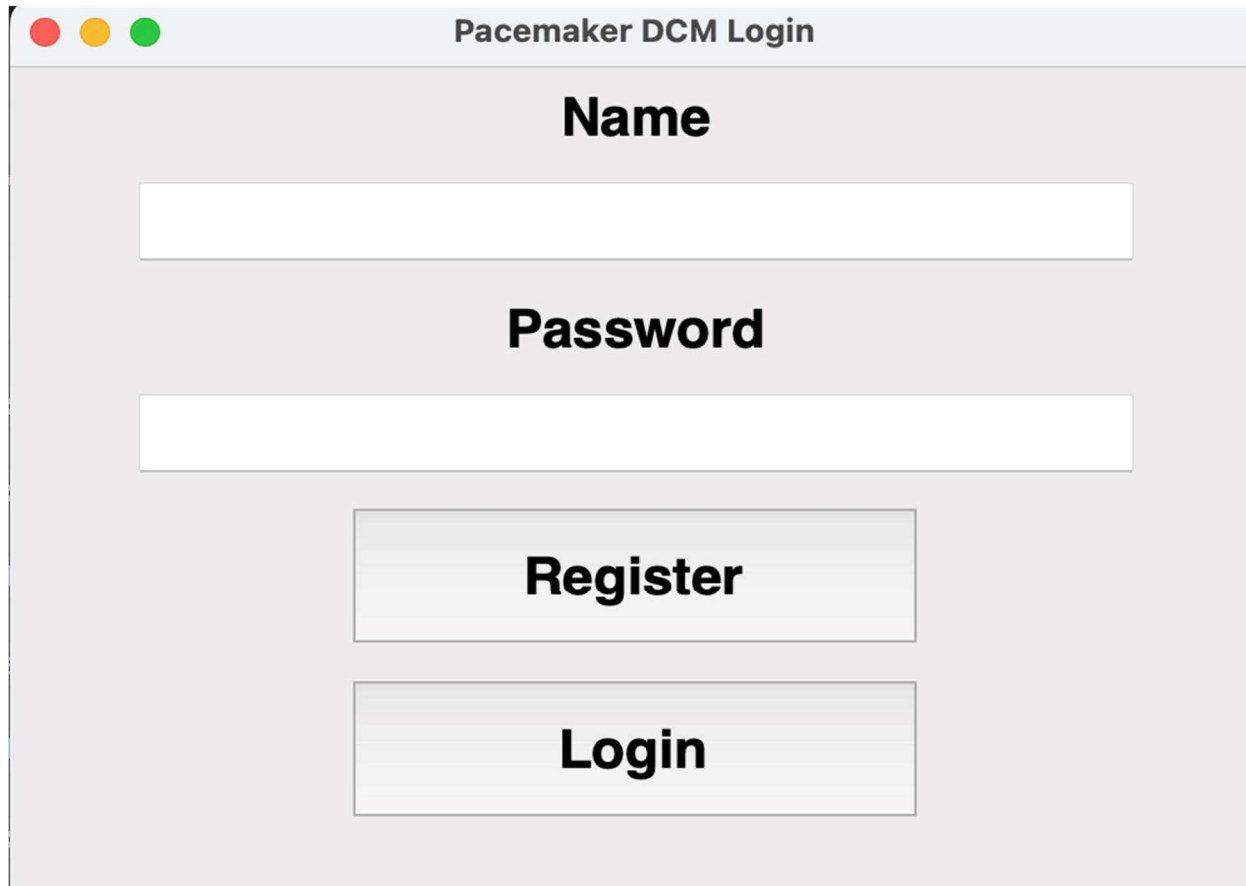
To handle user information and parameters, **json** was selected due to its simplicity in managing data and compatibility with text-based data storing. **Json** allowed for easy retrieval and saving of parameters and user data, which is crucial for managing user profiles and configuration preferences straightforwardly.

The **tkFont** module was chosen to allow customization of font styles and sizes, an important feature for creating a clear, readable interface that is enhanced visually. The ability to adjust font appearance and size ensures that users can easily navigate the DCM's interface.

Finally, **bcrypt** is used for securely storing user passwords. This library applies robust hashing algorithms to protect user credentials, providing an additional layer of security to the DCM application. By integrating bcrypt, we ensure that sensitive information, such as passwords, is securely managed, reducing the risk of unauthorized access.

### 3.3.2 Welcome Page UI

The welcome page was designed to meet all the requirements of the page located in section 3.2.1.

A screenshot of a macOS-style window titled "Pacemaker DCM Login". The window has a light gray title bar with three colored window control buttons (red, yellow, green) on the left. The main content area has a light gray background. It features two input fields: the first is labeled "Name" in bold black text, and the second is labeled "Password" in bold black text. Below the password field are two buttons: "Register" and "Login", both in bold black text. The buttons are light gray with a subtle gradient and a thin black border.

**Pacemaker DCM Login**

**Name**

**Password**

**Register**

**Login**

Figure 2222: Welcome Page

The welcome page is shown above which includes the two features to either sign in or login.

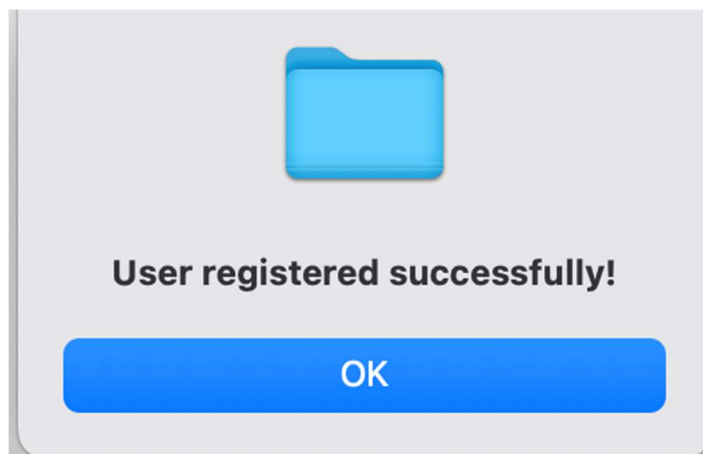


Figure 2323: Registration Confirmation

When the user enters a new username which has not been registered by another user and a password of at least 1 character, their new profile will be registered within the Json data file. This will only be successful when less than 10 users have been registered, as this variable holds a maximum value of 10.

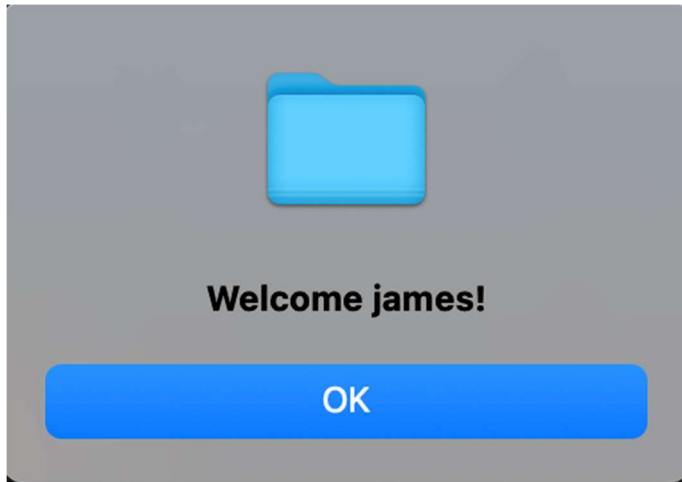


Figure 24 24: Welcome confirmation

Figure 13 displays the pop-up confirmation after a user logs into the application. This is applicable when a user has previously registered and has entered their correct username and password.

```
def load_users(self):
    try:
        with open(USERS_FILE, 'r') as f:
            return json.load(f)
    except FileNotFoundError:
        return {}

def save_users(self):
    with open(USERS_FILE, 'w') as f:
        json.dump(self.users, f, indent=4)
```

Figure 25 25: Login/Register functions

The above shows the code used to program the load and save users functions. The users are stored in a Json file.

```

"james": {
  "password": "$2b$12$iQpJwGeX5GnUiQY5NeGf.eWD3lfwb30W43ron1oARcmCrnCxlLux6",
  "mode": "VVI",
  "parameters": {
    "Lower Rate Limit": 175,
    "Upper Rate Limit": 176,
    "Atrial Amplitude": 2.9,
    "Atrial Pulse Width": 1.5,
    "Ventricular Amplitude": 2.7,
    "Ventricular Pulse Width": 1.1,
    "VRP": 253,
    "ARP": 196
  }
}

```

Figure 26: User Data Json File

The Json file above is used to store users' data, including the programmed parameters and the selected mode. These parameters are repopulated every time the user logs in and can be changed using the save button on the main page. The password is hashed using Bcrypt, using the password function below.

```

def hash_password(self, password):
    salt = bcrypt.gensalt()
    hashed = bcrypt.hashpw(password.encode('utf-8'), salt)
    return hashed.decode('utf-8')

def check_password(self, hashed_password, password):
    return bcrypt.checkpw(password.encode('utf-8'), hashed_password.encode('utf-8'))

```

Figure 27: Password Encryption Functions

```

def register_user(self):
    name = self.entry_name.get()
    password = self.entry_password.get()

    if name in self.users:
        messagebox.showerror("Error", "User already exists!") #checking if the username has already been registred
        return

    if len(self.users) >= MAX_USERS:
        messagebox.showerror("Error", "Maximum user limit reached!") #checking if their is space for a user
        return

    if name and password:
        self.users[name] = {
            'password': self.hash_password(password), #if both are valid, we peform the following
            'mode': 'A00'
        }
        self.save_users()
        messagebox.showinfo("Success", "User registered successfully!") #appending the new user to the Json
        self.clear_entries() #clearing the registration fields
    else:
        messagebox.showerror("Error", "Please enter both name and password.") #otherwise error

def login_user(self):
    name = self.entry_name.get()
    password = self.entry_password.get()

    if name in self.users and self.check_password(self.users[name]['password'], password): #using the unhashing function
        messagebox.showinfo("Success", f"Welcome {name}!") #the inputted passcode
        self.open_pacemaker_interface(name) #if valid open
    else:
        messagebox.showerror("Error", "Failed Login Attempt.") #otherwise error

```

Figure 2828: User Login/Registration Functions

Both the login and register functions utilize the hash and check password functions to encode and decrypt the user's password to promote privacy and security, one of the requirements of the DCM.

### 3.3.3 Parameter Interface

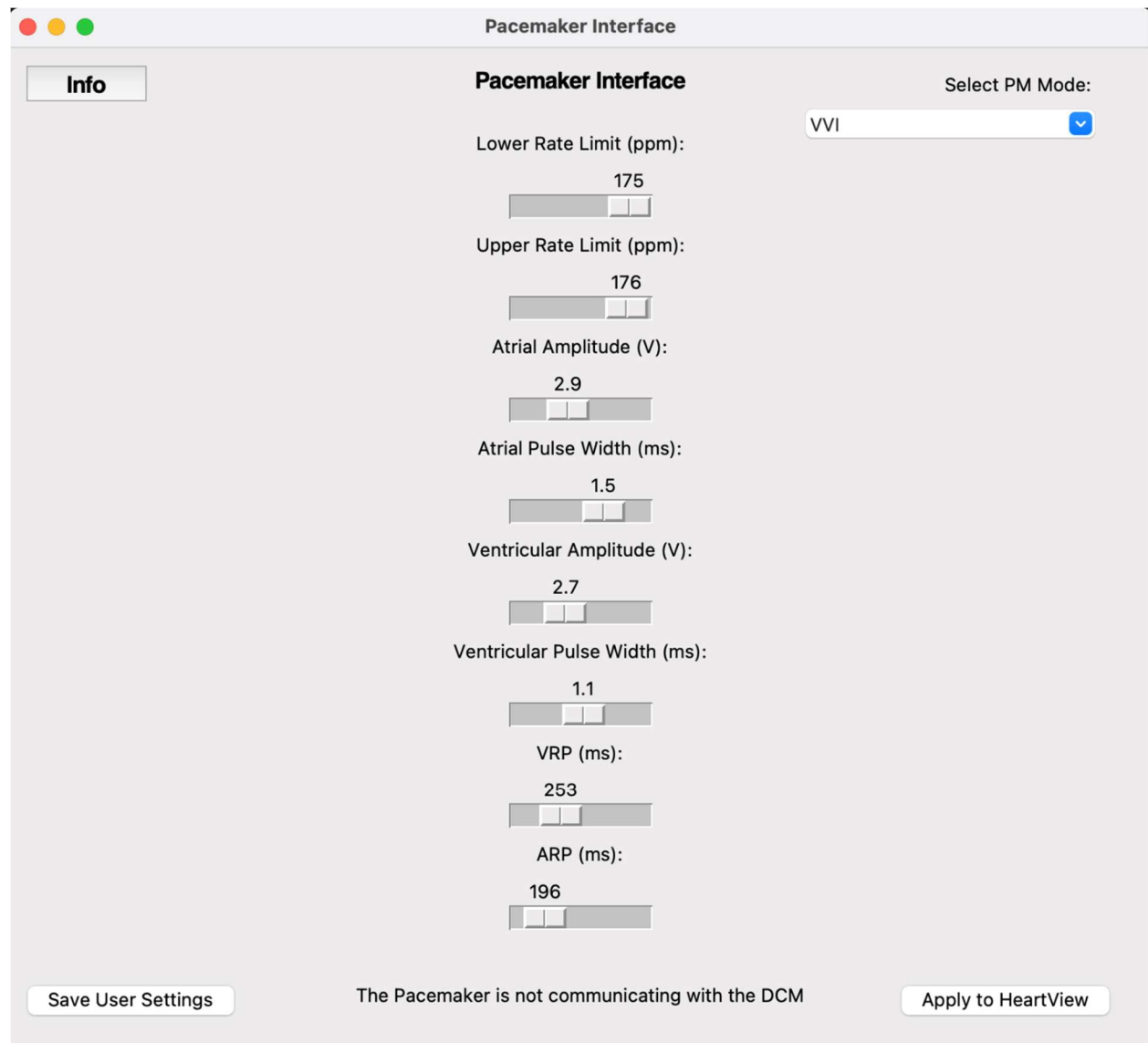


Figure 29: Main Interface

The above provides a clear view of the parameter interface, this being the main interface of the pacemaker UI. It includes slides for all the required parameters, only allowing the user to select an input within the required range. The sliders were programmed using a function to create each slider automatically promoting modularity within the code. The possibility of the user

accidentally setting the upper rate limit below the lower rate limit is evident therefore a function was added to prevent this, after this error was encountered through testing. The function can be found below, with comments present to briefly explain the function. It can also be seen that a message is displayed at the bottom of the application, telling the user whether or not the DCM is currently communicating with the pacemaker, as set in the DCM requirements.

```
def check_rate_limits(self, *args):
    lower_limit = self.sliders["Lower Rate Limit"].get() #retrieving the lower limit value
    upper_limit = self.sliders["Upper Rate Limit"].get() #retrieving the upper limit value

    if upper_limit < lower_limit:                #if upper less than lower, move upper to be one hgiher than lower
        self.sliders["Upper Rate Limit"].set(lower_limit + 1)
```

Figure 30: Rate limit verification function

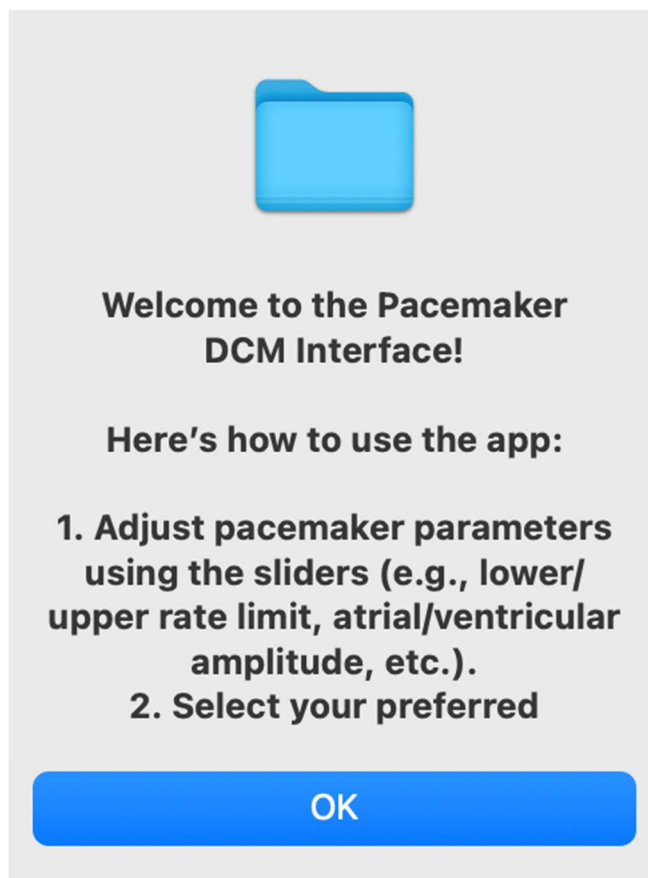


Figure 31: Information pop-up

The figure above presents the output of the info button found in the top right corner of the design. This feature was added after testing presented problems with new users who were unsure

of how the application functioned. It provides insight as to how the parameters can be manipulated and how the mode can be saved.

### 3.3.4 Heartview Connection Interface

The button shown in the bottom right of the screen was added in anticipation of assignment 2. It presents the user with a pop-up window displaying a message, indicating that once completed the Egram data will be present on this screen.

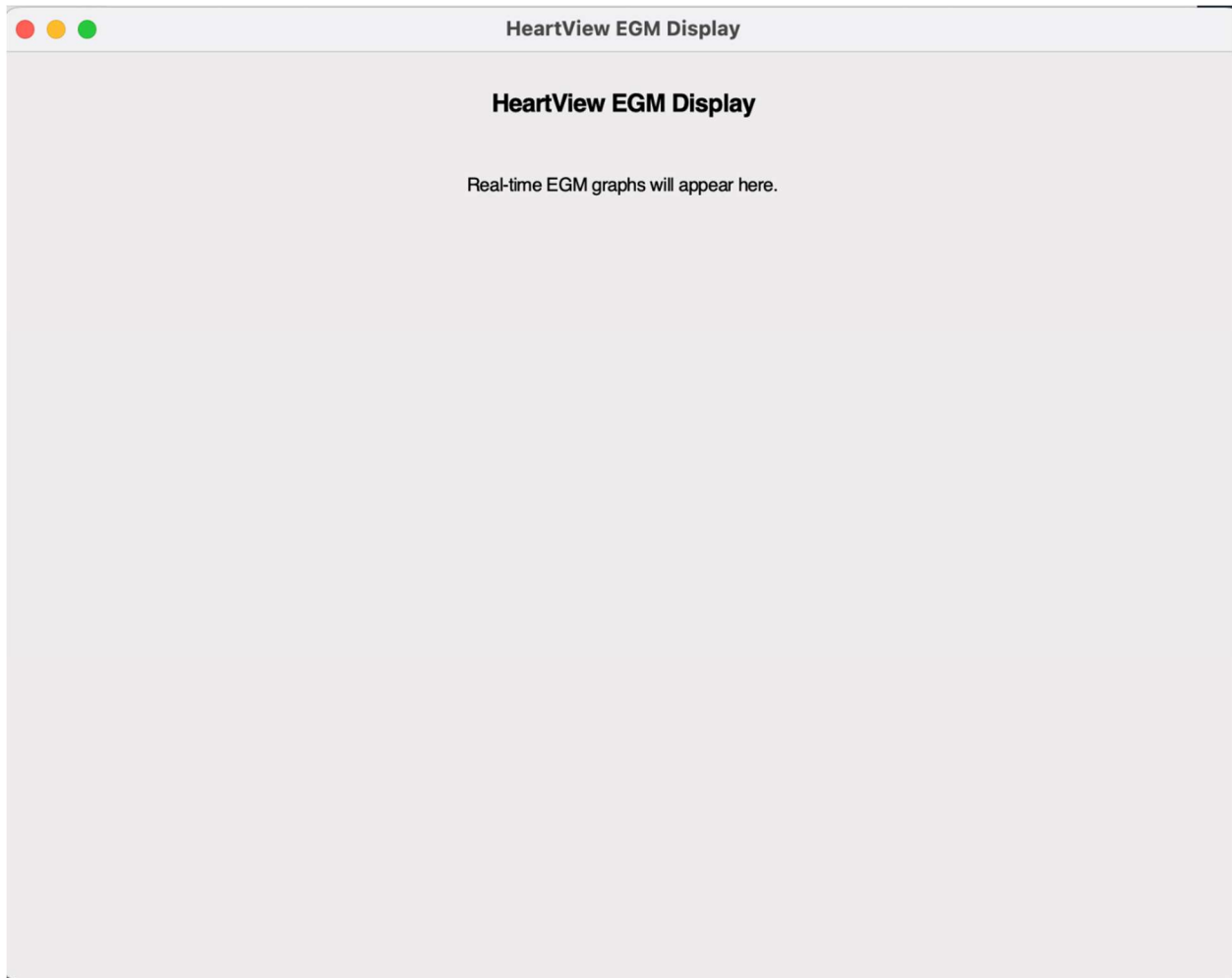


Figure 32: EGM window

```

def open_pacemaker_interface(self, name):
    self.root.withdraw() #removing the welcome screen
    pacemaker_window = tk.Toplevel(self.root)
    pacemaker_window.title("Pacemaker Interface")
    pacemaker_window.geometry("800x700")

    def on_close():
        self.root.destroy()
        pacemaker_window.destroy() # Close the pacemaker window

    pacemaker_window.protocol("WM_DELETE_WINDOW", on_close) #bindin gthe exit button on the screen to closing the application

    title_font = tkFont.Font(family="Helvetica", size=16, weight="bold") #creating the title font and title
    tk.Label(pacemaker_window, text="Pacemaker Interface", font=title_font).pack(pady=10)

    slider_frame = tk.Frame(pacemaker_window) #creating the frame where all the sliders will go
    slider_frame.pack(pady=10, anchor="center")

```

Figure 33: Main interface code 1

```

self.sliders = {} #creating sliders for all the paramters using the create_slider function
self.sliders["Lower Rate Limit"] = self.create_slider(slider_frame, "Lower Rate Limit (ppm):", 30, 175, callback=self.check_rate_limits)
self.sliders["Upper Rate Limit"] = self.create_slider(slider_frame, "Upper Rate Limit (ppm):", 60, 180, callback=self.check_rate_limits)

self.sliders["Atrial Amplitude"] = self.create_slider(slider_frame, "Atrial Amplitude (V):", 0.5, 7.0, resolution=0.1)
self.sliders["Atrial Pulse Width"] = self.create_slider(slider_frame, "Atrial Pulse Width (ms):", 0.1, 2.0, resolution=0.1)
self.sliders["Ventricular Amplitude"] = self.create_slider(slider_frame, "Ventricular Amplitude (V):", 0.5, 7.0, resolution=0.1)
self.sliders["Ventricular Pulse Width"] = self.create_slider(slider_frame, "Ventricular Pulse Width (ms):", 0.1, 2.0, resolution=0.1)
self.sliders["VRP"] = self.create_slider(slider_frame, "VRP (ms):", 150, 500)
self.sliders["ARP"] = self.create_slider(slider_frame, "ARP (ms):", 150, 500)

tk.Label(pacemaker_window, text="Select PM Mode:").place(relx=0.95, rely=0.02, anchor='ne') #dropdown for slecting the mode
mode_options = ["A00", "V00", "AAI", "VVI"]
mode_dropdown = ttk.Combobox(pacemaker_window, values=mode_options, state="readonly")

if name in self.users: #if name is in users we set their parameters as saved
    saved_parameters = self.users[name].get('parameters', {})
    for key in self.sliders:

```

Figure 34: Main interface code 2

```

        if key in saved_parameters:
            self.sliders[key].set(saved_parameters[key])
        mode_dropdown.set(self.users[name]['mode'])
    else:
        mode_dropdown.set("A00")

    mode_dropdown.place(relx=0.95, rely=0.06, anchor='ne')

    #buttons for all the functions previouslt defined
    tk.Button(pacemaker_window, text="Info", command=self.show_info, font=title_font, width=6, height=1).place(x=10, y=10)
    tk.Button(pacemaker_window, text="Save User Settings", command=lambda: self.save_mode(name, mode_dropdown.get())).place(x=10, y=650)
    tk.Label(pacemaker_window, text="The Pacemaker is not communicating with the DCM").place(x=240, y=650)
    tk.Button(pacemaker_window, text="Apply to HeartView", command=self.show_heartview).place(x=620, y=650)

Initialize and run the application|
cemakerApp()

```

Figure 35: Main interface code 3

The three figures above present the main logic for the interface. It shows the removal of the login window and how it is closed. It then moves into the formatting of the visuals the user



will see including the sliders for all the parameters of the requirements. It also shows the creation of the buttons shown on the interface and how they are positioned in specific positions to enhance the user experience. The rest of the functions are explained below in section 3.4.

## 3.4 Module Overview

### 3.4.1 Purpose of the Module

The PacemakerApp module is designed to provide a graphical user interface (GUI) for managing user logins and pacemaker settings. It allows users to register, log in, adjust pacemaker parameters, and view a simulated real-time Electrogram (EGM) display. The application uses secure password storage and limits the number of registered users.

```
# Constants
MAX_USERS = 10
USERS_FILE = 'users.json'

class PacemakerApp:
> def __init__(self):~
> def load_users(self):~
> def save_users(self):~
> def hash_password(self, password):~
> def check_password(self, hashed_password, password):~
> def register_user(self):~
> def login_user(self):~
> def clear_entries(self):~
> def show_info(self):~
> def check_rate_limits(self, *args):~
> def create_slider(self, frame, label_text, from_val, to_val, resolution=1, callback=None):~
> def save_mode(self, name, selected_mode):~
> def show_heartview(self):~
> def open_pacemaker_interface(self, name):~

# Initialize and run the application
PacemakerApp()
```

Figure 36: List of functions

### 3.4.2 List of Public Functions

1. **\_\_init\_\_(self)**
  - **Parameters:** None
  - Initializes the application, sets up the main window, and loads existing users.
2. **load\_users(self)**
  - **Parameters:** None
  - Loads user data from a JSON file.

3. **save\_users(self)**

- **Parameters:** None
- Saves user data to a JSON file.

4. **hash\_password(self, password)**

- **Parameters:**
  - password: (str) The plain text password to be hashed.
- Returns the hashed password.

5. **check\_password(self, hashed\_password, password)**

- **Parameters:**
  - hashed\_password: (str) The hashed password stored for the user.
  - password: (str) The plain text password to check against the hashed password.
- Returns a boolean indicating whether the password matches.

6. **register\_user(self)**

- **Parameters:** None
- Handles user registration logic.

7. **login\_user(self)**

- **Parameters:** None
- Handles user login logic.

8. **clear\_entries(self)**

- **Parameters:** None
- Clears the username and password entry fields.

9. **show\_info(self)**

- **Parameters:** None
- Displays a message box with usage instructions.

10. **check\_rate\_limits(self, \*args)**

- **Parameters:** \*args
- Validates that the upper rate limit is greater than the lower rate limit.

11. **create\_slider(self, frame, label\_text, from\_val, to\_val, resolution=1, callback=None)**

- **Parameters:**
  - frame: (tk.Frame) The frame in which to create the slider.

- **label\_text:** (str) The label for the slider.
  - **from\_val:** (int/float) The minimum value for the slider.
  - **to\_val:** (int/float) The maximum value for the slider.
  - **resolution:** (int/float) The increment step for the slider.
  - **callback:** (function) A callback function to call when the slider value changes.
  - Returns the created slider widget.
12. **save\_mode(self, name, selected\_mode)**
- **Parameters:**
    - **name:** (str) The name of the user.
    - **selected\_mode:** (str) The selected pacemaker mode.
  - Saves the user's selected mode and parameters.
13. **show\_heartview(self)**
- **Parameters:** None
  - Opens a new window for displaying EGM data.
14. **open\_pacemaker\_interface(self, name)**
- **Parameters:**
    - **name:** (str) The name of the logged-in user.
  - Opens the pacemaker settings interface.

### 3.4.3 Black Box Behavior

1. **\_\_init\_\_:** Initializes the GUI and loads user data.
2. **load\_users:** Reads the user data from users.json and returns it as a dictionary; returns an empty dictionary if the file does not exist.
3. **save\_users:** Writes the current user data to users.json.
4. **hash\_password:** Takes a plain text password, generates a hash, and returns it as a string.
5. **check\_password:** Compares a plain text password with a hashed password; returns True if they match, False otherwise.
6. **register\_user:** Collects user data from input fields and registers a new user if conditions are met; displays error messages for invalid scenarios.

7. **login\_user**: Validates user credentials and opens the pacemaker interface upon successful login; displays error messages for failed attempts.
8. **clear\_entries**: Empties the username and password input fields.
9. **show\_info**: Displays a message box containing instructions for using the application.
10. **check\_rate\_limits**: Ensures the upper limit is greater than the lower limit; adjusts if necessary.
11. **create\_slider**: Creates a slider with specified attributes and returns it; calls a callback if provided.
12. **save\_mode**: Saves the selected mode and current parameters for the user; confirms success through a message box.
13. **show\_heartview**: Opens a new window with a placeholder for EGM data visualization.
14. **open\_pacemaker\_interface**: Launches the pacemaker settings interface, populating sliders with saved parameters if available.

### 3.4.4 Global Variables

- **self.users**: A dictionary that stores user data, with usernames as keys and their corresponding passwords and settings as values.

- **Structure:**

```
json
Copy code
{
  "username": {
    "password": "hashed_password",
    "mode": "AOO",
    "parameters": {
      "Lower Rate Limit": value,
      "Upper Rate Limit": value,
      ...
    }
  },
}
```

```
...  
}
```

### 3.4.5 Private Functions

The current module does not contain explicitly private functions, as all defined functions are public. However, functions like `load_users`, `save_users`, and `hash_password` could be considered private in the context of their usage within the module.

### 3.4.6 Internal Behavior

1. **`__init__`**
  - Initializes `self.users` by calling `load_users()`.
  - Sets up the main application window with a title and dimensions.
  - Creates input fields for username and password and buttons for registration and login.
2. **`load_users`**
  - Attempts to read `users.json`:
    - On success, loads user data into `self.users`.
    - On failure (file not found), returns an empty dictionary.
3. **`save_users`**
  - Serializes `self.users` and writes it to `users.json`.
4. **`hash_password`**
  - Takes a plain text password, generates a salt using `bcrypt`, hashes the password, and returns the hashed string.
5. **`check_password`**
  - Compares a provided plain text password with the stored hashed password, returning `True` if they match.
6. **`register_user`**
  - Retrieves user input from entry fields.
  - Validates if the user already exists and checks the maximum user limit.
  - If validation passes, hashes the password, saves the user, and displays success/error messages.

## 7. **login\_user**

- Collects username and password from input fields.
- Validates credentials:
  - If valid, opens the pacemaker interface.
  - If invalid, shows an error message.

## 8. **clear\_entries**

- Empties the content of the username and password entry fields.

## 9. **show\_info**

- Constructs an information message and displays it in a message box.

## 10. **check\_rate\_limits**

- Monitors the slider values for lower and upper rate limits.
- Adjusts the upper limit if it is lower than the lower limit.

## 11. **create\_slider**

- Creates a slider with specified attributes.
- If a callback function is provided, associates it with the slider.
- Returns the created slider widget.

## 12. **save\_mode**

- Updates the user's mode and saves the current parameters to self.users.
- Calls save\_users to persist changes and shows a success message.

## 13. **show\_heartview**

- Creates a new window for EGM data visualization, currently containing placeholder text.

## 14. **open\_pacemaker\_interface**

- Hides the main window and sets up a new window for pacemaker settings.
- Initializes sliders and a mode dropdown, populating with saved settings if available.
- Adds buttons for showing info, saving settings, and applying to HeartView.

### 3.5 Testing and Results

Testing was performed by manually verifying the user registration, login process, and parameter inputs. Each mode was selected, and the associated parameters were adjusted and stored locally. Future integration with the pacemaker hardware is planned for later stages.

Test Description	Input	Expected Result	Actual Result	Pass/Fail	Comments/Changes Needed
Register a new user with expected input	Name: "Alice", Password: "password123"	User registered successfully message displayed and moved on	User registered successfully message displayed and moved on	Pass	N/A
Attempt to register the same user twice	Name: "Alice", Password: "newpassword"	Error message "User already exists!" displayed.	Error message "User already exists!" displayed.	Pass	N/A
Register a user when user limit is reached	Name: "User10", Password: "pass123"	Error message "Maximum user limit reached!" displayed.	Error message "Maximum user limit reached!" displayed.	Pass	N/A
Attempt to register a user without a name	Name: "", Password: "password"	Error message "Please enter both name and password."	Error message "Please enter both name and password."	Pass	N/A

Attempt to login with correct credentials	Name: "Alice", Password: "password123"	Welcome message "Welcome Alice!" displayed.	Welcome message "Welcome Alice!" displayed.	Pass	N/A
Attempt to login with empty fields	Name: "", Password: ""	Error message "Please enter both name and password."	Error message "Please enter both name and password."	Pass	N/A
Check rate limits functionality	Lower Limit: 60 Upper Limit: 50	Upper limit should automatically set to 61.	Upper limit set to 60.	Fail	Update the check_rate_limits logic to enforce limits.
Attempt to save user mode with empty mode value	Mode: ""	Error message "Mode cannot be empty."	Mode saved	Fail	Ensure mode selection cannot be left blank
Set lower limit at 0 bpm	Lower Limit: 0, Upper Limit: 100	Error message "Lower limit cannot be zero."	Limits set	Fail	Establish reasonable minimums for limits

Table 14: DCM Testing



## **3.6 Potential Changes and Future Considerations**

### **3.6.1 Requirements Changes**

In the future, there are many changes that will be required. Due to the increase in Simulink modes, parameters, and other factors, there will be a rise in the number of requirements for the DCM in the future. Since the DCM will be required to communicate directly with the pacemaker for assignment 2, many requirements which were not present previously will come into play.

The user interface will be required to be capable of visually indicating when telemetry is lost due to the device being out of range. This is crucial because if this was ever presented in a real-life situation, quick action would be required to ensure the connection is not lost and the pacemaker can still perform its job. Along with this, the UI must also be capable of visually indicating when telemetry is lost due to noise. For the same reasons as before this is equally as crucial.

The DCM will need to include an About function to provide essential system details, displaying the application model and version, DCM serial number, and the institution's name. It will also need to contain a Set Clock function that enables users to configure the device's date and time for accurate logging. Along with this, a New Patient function, where a new device can be connected and integrated without requiring the application to restart, streamlining patient management. Lastly, a Quit function will be required to allow for a telemetry session to be ended at any time, without causing any issues and freeing up system resources for other patients.

The DCM will be required to offer user-requested reports for parameters and status monitoring, including the Bradycardia Parameters, Temporary Parameters, Implant Data, Threshold Test Results, Measured Data, Marker Legend, Session Net Change, and a comprehensive Final Report that combines key telemetry and device data.

In addition, it must offer diagnostic reports, such as the Rate Histogram and Trending reports for in-depth bradycardia analysis. Each report will need to include a standardized header

displaying the institution name, report timestamp, device and DCM serial numbers, and software model and version for consistent identification across documentation.

The DCM will need to enable strip chart recording, to allow real-time visualization and recording of ECG data. The system will need to be able to support up to three Real time traces (2 Telemetered, 1 Surface ECG), as well as an annotation for display of event markers. It will use its internal strip chart recorder to perform these actions. It must also be capable of printing real-time telemetered data and a surface ECG.

### **3.6.2 Design Changes**

As a DCM is constantly changing, there will always be design changes that could be made to improve user experience and the application's ability to complete its job of providing safety and security to its users.

Currently, user data is stored in a JSON file, which can limit the scalability and security of the system. A future design enhancement could involve integrating a larger database solution, such as SQL, to handle larger data volumes and support more complex data streams. This upgrade would improve data management and improve performance, particularly if the DCM has a growing user base. A database solution could also allow multiple users to have access to the platform at once, which would be essential for expanding the application's capabilities.

When incorporating EGM data into the system, creating a dedicated page for its visualization would provide users with real-time insights into the patient's cardiac rhythms. The page currently present for the improvements in this area could feature customizable graphs, alerts, or markers for critical heartbeat events, allowing users to analyze EGM data directly within the application. These features would not only enhance the DCM's usability but also support better decision-making for doctors and patients.

To continuously improve the user experience, collecting feedback through surveys would be beneficial. By gathering insights on which areas users find challenging, these areas can be prioritized to directly address weak points. This feedback-driven approach allows for improvement in areas that may be looked over by the team to be targeted, improving accessibility, and overall usability. Implementing these changes would align the application's evolution with user needs, making it more intuitive and efficient for a wider range of users.

In future iterations of the DCM, implementing more OOP (object-oriented programming) would be crucial for improving code organization and maintainability. By dividing newly developed code into classes and reusable components, the application can become increasingly modular, making future code easier to implement. An OOP approach also helps with testing as it can be easier to create isolated, modular tests for each component. OOP would also reduce redundancy and coupling, some factors that are important to limit for the code to be understandable for other professionals, aside from just programmers.

### 3.7 DCM Development History

Date	Changes
September 29, 2024	<p>Initialized the DCM application using the Tkinter library.</p> <p>Defined constants for maximum users (MAX_USERS = 10) and the user data file (USERS_FILE = 'users.json').</p> <p>Created the PacemakerApp class, handling user login and registration.</p>
October 4, 2024	<p>Created the login interface with "Name" and "Password" fields.</p> <p>Configured "Register" and "Login" buttons to call register_user and login_user methods, respectively.</p>
October 7, 2024	<p>Implemented load_users and save_users functions to manage user data in JSON format.</p> <p>Verified correct functionality for user registration, including unique username checks and the maximum user limit.</p>
October 10, 2024	<p>Set up the main pacemaker interface to display adjustable parameters, including rate limits, atrial and ventricular amplitudes, and pulse widths. Added sliders with real-time validation (check_rate_limits) to ensure logical parameter relationships.</p> <p>Integrated mode selection with a dropdown menu, defaulting to "AOO." Enabled dynamic saving of the selected mode and slider parameters per user profile using the save_mode function.</p>
October 16, 2024	<p>Created the show_info function to guide users in adjusting pacemaker parameters.</p> <p>Added a "HeartView" visualization placeholder for displaying EGM data as required, confirming functionality for parameter retrieval, display, and saved settings reloading on subsequent logins.</p>
October 22, 2024	<p>Validated the complete functionality of the DCM application, ensuring user access and parameter saving/recall features work seamlessly.</p> <p>Added bcrypt for secure password hashing; implemented functions hash_password and check_password.</p>

Table 15: DCM Design Log

## **4. Conclusion**

This document outlines the design, development, and testing of the pacemaker modes and DCM interface. The system meets the current requirements, and further work will be carried out in future assignments to integrate the DCM with the pacemaker hardware and ensure full functionality.