
MECHTRON 3K04: Software Development Assignment 1 PACEMAKER Project (Group 19)

Himanshu Singh (singh41)
Mathew Galuszka (galuszkm)
Shaan Suthar (suthas2)
Shivan Gaur (gaurs5)
Varun Kothandaraman (kothandv)
Christopher Nazarian (nazariac)

Table of Contents

Revision History	4
Introduction	5
Planning & Research	6
Preliminary	6
Pacemaker	6
Device Controller Monitor (DCM)	6
DCM Comparison – GUI Breakdown.....	6
DCM Comparison – Data Management.....	8
DCM Comparison – Serial Communication	9
Requirements & Specifications.....	9
Current Pacemaker Requirements:.....	9
Future Pacemaker Requirements:.....	10
DCM Requirements.....	10
Design	12
Pacemaker Design	12
AOO	12
VOO	13
AAI.....	13
VVI.....	14
Mode Switching	14
Hardware Hiding	15
Challenges	17
DCM Design	18
Repository	18
Imported Libraries	19
Classes Descriptions.....	19
Functions	19
Problems and Challenges	21
GitHub Revision History	Error! Bookmark not defined.
Final Screenshots.....	22
Monitored Variables & Constants	25
AOO	25
VOO	25
AAI.....	26

VVI.....	26
Decision Tables.....	28
Pacemaker	28
<i>AOO</i>	28
<i>VOO</i>	28
<i>AAI</i>	28
<i>VVI</i>	28
DCM.....	28
Testing.....	30
AOO & VOO Testing	30
AAI & VVI Testing.....	30

Revision History

<i>Revision</i>	<i>Description</i>	<i>Date (MM/DD/YYYY)</i>
<i>A1</i>	Assignment 1 Pacemaker design, Device Controller Monitor and Design Documentation Submission. Authors: Himanshu Singh, Mathew Galuszka, Shaan Suthar, Shivan Gaur, Varun Kothandaraman, Christopher Nazarian	10/15/2023

Introduction

Pacemakers are electrical devices that provide the heart with electrical pulses through the subclavian vein to actuate the right atrium and ventricle to aid with a patient's irregular heartbeat. This process is done to mainly regulate the electrical conduction system of the cardiac cycle to allow for proper blood flow and to maintain the patient's overall health.

The following documentation provides a look into the design and development process that we utilized to create our pacemaker through MATLAB Simulink while being simulated alongside Heartview to confirm functionality of our state flow diagrams.

For the pacemaker to properly adapt to a patient's cardiac cycle, a Device Controller Monitor (DCM) was designed to allow for the alteration of pacemaker behavior and for the output of important cardiac cycle data.

Our DCM implementation was tasked with storing the information of up to ten registered users as well as each user being able to modify the default parameters of the pacemaker itself.

Planning & Research

Preliminary

Before beginning with either the DCM or Simulink, a GitHub repository was created to allow for access and collaboration on files between group members. It was decided that splitting into two subgroups—one to handle the Simulink and the other the DCM—would be the most efficient way of tackling both components in the given time frame. Understanding the requirements was also key, therefore all the critical documentation, including the assignment, pacemaker shield and pacemaker docs were reviewed prior to any the start of any physical work.

Pacemaker

To help with debugging, a decision was made to create each pacing mode in its own MATLAB Simulink file, and then combine all of the modes into one single file when they were all functioning correctly. In retrospect this also helped with compile times and overall organization of code for testing.

Device Controller Monitor (DCM)

Due to the flexibility in the DCM Requirement Specifications regarding the selection of technologies, it is imperative to thoroughly evaluate all available options before making a decision on the implementation. To facilitate this technology selection process, the DCM has been categorized into three distinct areas: GUI display, data management, and serial communication. Each category's potential implementations were scrutinized and assessed to ascertain their suitability for the project.

DCM Comparison – GUI Breakdown

Python → Tkinter: The Tkinter library in Python is an excellent choice for developing a straightforward, locally run pop-up application, particularly when the project's interface requirements are not overly complex. Additionally, Python offers another significant advantage for this project due to the availability of a well-documented library for facilitating serial communication between the pacemaker and Python, ensuring smooth and efficient data exchange.

Furthermore, opting for Python provides the added benefit of capitalizing on the collective expertise of the project team. Every team member is guaranteed to have experience in programming with Python, as it was introduced during their first year in the Engineering program or the Integrated Biomedical Engineering and Health Sciences (IBioMed) program. This shared proficiency in Python will streamline collaboration and development, making it an ideal choice for our project.

Java → Swing: Within our team of six members, two possess a wealth of experience in developing intricate video games and applications using the Java Swing library. Leveraging Java for our project offers several compelling advantages. Firstly, it allows us to harness its robust

object-oriented capabilities, which can greatly enhance the development process. Additionally, Java follows the Model-View-Controller (MVC) paradigm, which empowers us to create a more adaptable and flexible User Interface (UI).

However, it's essential to acknowledge a potential drawback. Our Teaching Assistants (TAs) lack familiarity with serial communication in Java, a technical aspect that may not be directly required for Assignment 1 but holds future significance. Navigating the intricacies of a syntax-heavy language like Java without the guidance of teaching faculty could pose challenges down the road. Thus, while Java offers notable advantages, we must consider the potential hurdles of limited TA support when making our decision.

HTML/CSS/JavaScript: The most widely recognized and preferred languages for developing frontend applications are the cornerstone web development languages: HTML, CSS, and JavaScript. Remarkably, three out of our six group members possess prior experience working with these frameworks. This foundation in these languages bestows upon us a considerable advantage.

Nevertheless, it's crucial to acknowledge a potential challenge. One significant drawback of relying on HTML, CSS, and JavaScript is ensuring that our Teaching Assistants (TAs) can comprehensively understand our code. This challenge primarily stems from the intricacies of commenting CSS code, which can often be perplexing and problematic. As such, while these web development languages offer substantial benefits, it's essential to be mindful of the potential complexity in code comprehension for our TAs.

Decision:

In our pursuit of Graphical User Interface (GUI) development, our team arrived at the deliberate choice of harnessing the power of TKinter in tandem with the Python programming language. This decision was underpinned by a multitude of compelling factors, as previously deliberated. Firstly, TKinter emerged as the ideal candidate for crafting streamlined, locally executed pop-up applications, perfectly aligning with the simplicity our project required, sparing us from the complexities of an overly intricate interface. Moreover, Python's innate strengths in object-oriented programming and adherence to the Model-View-Controller (MVC) paradigm bestowed upon us a flexible canvas to paint our UI upon.

What truly swayed our decision, however, was the prospect of future-proofing our serial communication capabilities, thus alleviating a considerable burden from our workload. The realization that we wouldn't need to search externally for these capabilities in the future proved to be the decisive factor. While we acknowledged the absence of bonus points for extravagant GUI implementations, we opted for an aesthetic and functional GUI that exceeded the minimum requirements but stayed within reasonable bounds, at least for this initial assignment. This prudent approach ensured that our shared expertise, honed during our first-year experiences in Engineering and IBioMed programs, paved the way for a seamless collaborative development process. In sum, the amalgamation of these factors fortified our unwavering choice of TKinter and Python as our preferred technologies for GUI development.

DCM Comparison – Data Management

For our implementation, we considered various technologies and languages.

MongoDB: We could implement MongoDB using Python with MongoPy or JavaScript with Mongoose. MongoDB is a non-relational database known for its excellent documentation and relatively easy learning curve. It also offers supporting packages that make integration with Flask applications more convenient. However, the development team's experience with MongoDB was somewhat limited.

Firebase: We could implement Firebase using JavaScript with the Firebase SDK. Firebase is another non-relational database, known for its excellent documentation. However, it has a requirement that users must register an account with Google Apps and provide payment information, which could be a drawback for some.

SQL: For SQL databases, we would use Postgres and Sqlite3 with Python. SQL databases are relational, which aligns well with the data representation needs of our project. Using Sqlite3 with Python allows for direct integration into a Python project. As a result, SQL databases received a relatively high overall score of 7 out of 10, making them a strong contender for our implementation.

Locally Stored: An alternative approach, distinct from relying on an online data management technology, involves the local storage of all data and states within the computer running the DCM file. One method to achieve this is by saving the user's data in a text file on the local machine. While some may initially view this choice as a potential manifestation of laziness, the rationale behind it is, in fact, far more grounded in logic than it may seem. This decision is particularly pertinent given the nature of the project, which revolves around a pacemaker.

In the context of a pacemaker, mandating an internet connection for updating its settings can introduce significant risks in real-life scenarios. Therefore, it becomes considerably safer and more prudent to leverage the concept of encapsulation, wherein the data and the program coexist locally. This approach minimizes potential vulnerabilities associated with external dependencies, ensuring that the pacemaker's functionality remains robust and reliable, free from the inherent uncertainties of online connectivity.

Decision: Indeed, when we delve deeper into the considerations outlined earlier, the choice of local storage for data management becomes not only reasonable but also the most sensible course of action. The paramount reason, as previously discussed, is the inherent safety and reliability it offers in the context of a pacemaker project.

As a life-critical medical device, the pacemaker's operation demands an extra layer of caution. Requiring an internet connection to update its settings would introduce a grave level of risk into real-life scenarios. Connectivity issues, data breaches, or even the possibility of unauthorized access could jeopardize the pacemaker's functionality, potentially endangering the patient's life.

By opting for local data storage, we adhere to the principles of encapsulation, ensuring that the data and program operate in a self-contained environment, impervious to external disruptions. This approach is not a manifestation of laziness, as some might initially perceive it, but rather a strategic choice made to safeguard the integrity of the pacemaker and prioritize patient safety above all else.

Additionally, the usernames and passwords are encoded rather than stored as plain text. The password is hashed using the SHA-256 algorithm. This is a widely used and secure cryptographic hash function. The hashed password is then stored, enhancing the security of user credentials within the system. It's important to note that when users log in, their entered passwords are hashed using the same algorithm, and the resulting hash is compared to the stored hash for authentication. This approach ensures that the actual password is never stored or transmitted in an easily readable format. Therefore, by following this industry security best practice we can protect the patients' sensitive, personal, and private information.

In essence, when the project's unique circumstances and life-critical implications are factored in, the preference for local storage emerges as the most rational and responsible choice for data management.

DCM Comparison – Serial Communication

In Assignment 1, there is presently no imperative need for serial communication integration. Consequently, our DCM is presently devoid of the functionality to communicate with our Pacemaker device, resulting in inactive buttons and unresponsive displays.

Requirements & Specifications

Current Pacemaker Requirements:

For a pacemaker in permanent state, implement stateflows in Matlab Simulink for the following pacemaker modes: AOO (atrium paced, no chambers sensed, no response to sensing), VOO (ventricle paced, no chambers sensed, no response to sensing), AAI (atrium paced, atrium sensed, inhibition in response to sensing), VVI (ventricle paced, ventricle sensed, inhibition in response to sensing). The parameters required for a stateflow should be programmable and are listed below according to mode:

Table 1: Parameters for Pacemaker 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	

Hardware hiding is also required so that even if the pin map being used were to be altered, the stateflows would not change.

Future Pacemaker Requirements:

In the future, additional modes AOOR, VOOR, AAIR, VVIR all need to be implemented. The R stands for the inclusion of rate modulation on each mode. Some additional parameters also require implementation, including Rate smoothing, Hysteresis, and PVARP.

DCM Requirements

Before the start of designing our DCM, we first outlined the objective and purpose of our pacemaker. All of these requirements are outlined below in respect to their important and purpose:

Login Screen

Objective: Develop a secure and user-friendly login system that allows for account creation, access, and secure data storage of up to 10 users.

The system should support at least three primary functionalities: Logging in, creating an account, and securely storing user information. The system should provide feedback to the user regarding login success or failure upon interacting with UI elements as there should be clear error messages for common issues like incorrect passwords or already existing usernames. Ensure that there's a limit to the number of users, which is capped at 10.

Login Functionality:

1. The interface should have fields for the user to input their username and password.
2. Upon entering the correct username and password, users should be redirected to the next stage or main application interface.
3. If the password is incorrect, a clear notification should be displayed indicating the error.
4. Users should be given the option to reset their password or create a new account if they can't log in.
5. Additionally, the screen should display the time to the user.

Account Creation

1. There should be a clearly labeled option or button to create a new account.
2. During account creation, users should input at least a username and password.
3. Usernames should be unique; the system must notify the user if the chosen username already exists.
4. Provide feedback upon successful account creation or if there are any issues.

Secure Data Storage

1. User data storage should ensure minimal access while maintaining functionality.
2. Data should be stored in a "pacemaker" (assuming this is a secure storage option in the context given), which allows for continuous access regardless of external conditions.
3. All user data, especially passwords, should be encrypted using a strong and modern encryption algorithm.

Main App Interface

Objective: Design a user-friendly interface that allows users to navigate pacemaker modes, view and select parameters, and differentiate between regular and admin accesses. Additionally, the main app should have the ability to start and stop the program.

User Account Navigation:

1. Users must have the capability to log out and be redirected back to the login screen.

User Interface (UI) Functionality:

1. The UI should be clear to use and not cause confusion from the user.

Parameters Display:

1. The system should display four distinct pacemaker modes: AOO, VOO, AAI, and VVI.
2. Each mode should be accompanied by its related parameters as referenced in the "PACEMAKER Document: Table 6".

Mode Selection and Parameter Preview:

1. Users should have the option to select any of the pacemaker modes presented.
2. For every selected mode, users should be able to edit the corresponding parameters in the "PACEMAKER Documentation: Table 7"

Parameter Default Settings:

1. On accessing a pacemaker mode for the first time or after a reset, users should see the nominal values set for each parameter.

Additional Considerations:

1. Implement a clear hierarchy in displaying the pacemaker modes and their associated parameters to avoid user confusion.
2. Implement UI theme changes for accessibility in reading, as in light and dark mode

Design

Pacemaker Design

As mentioned in Planning and Research, the development of each mode occurred within individual files before being implemented. Due to the nature of how the pacing system operates, all modes share a dual state system where the only differences are how the transitions occur. These two states are called Pace and Refractory, and the following defines how these states operate in AOO and AAI.

In Refractory, Capacitor C22 needs to be charged and Capacitor C21 needs to be discharged. Because these operations are mutually exclusive, they were incorporated into a single state. According to the mandatory pin arrangement provided in the documentation, discharging C21 is accomplished by setting PACE_GND_CTRL and ATR_GND_CTRL high. Charging C22 is accomplished by setting PACE_CHARGE_CTRL high and setting PACING_REF_PWM to Atrium_Amplitude, which is the charging duty cycle. How the duty cycle is calculated is outlined in Hardware Hiding. In Pace, ATR_PACE_CTRL and PACE_GND_CTRL are set high to apply the pace to the heart. For VOO and VVI, the same variables are used with their names switched from atrium to ventricle. These changes include swapping ATR_GND_CTRL to VENT_GND_CTRL, Atrium_Amplitude to Ventricle_Amplitude, ATR_PACE_CTRL to VENT_PACE_CTRL, and Atrium_Pace_Width to Ventricle_Pace_Width. Additionally, during the pace state the LED on the pacemaker is set high to visualize the pacing of the heart. During the refractory state it is set low.

The following section details the transitions of each individual mode.

AOO

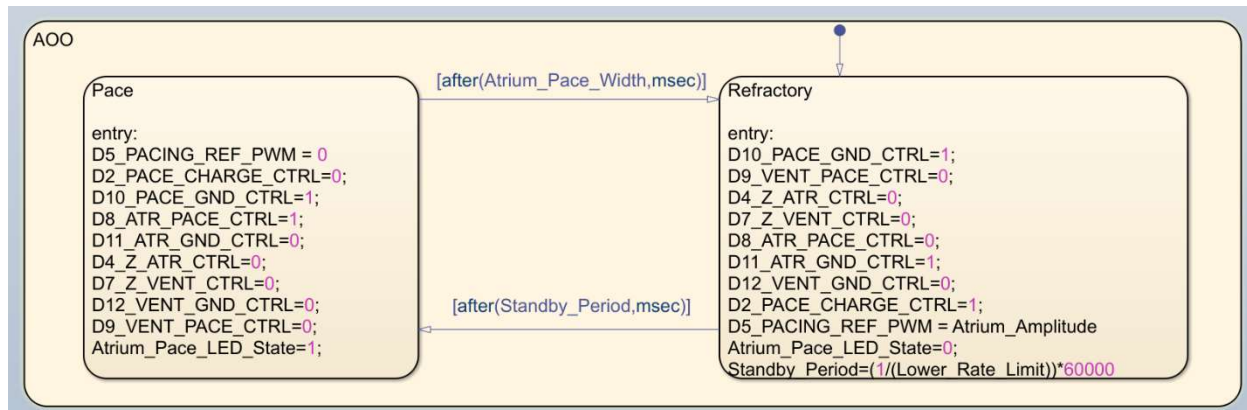


Figure 1 - AOO Stateflow

After all of the output pins are set in Refractory, the standby period is calculated by converting the Lower Rate Limit from BPM to a period in milliseconds. As this is a non-sensing mode, the standby period is determined solely by the LRL. The transition to Pace occurs after the calculated standby period. After all the output pins are set in Pace, the transition to refractory

occurs after the Atrium_Pace_Width, which is a programmable parameter that defines the pulse width of the pace sent to the heart.

VOO

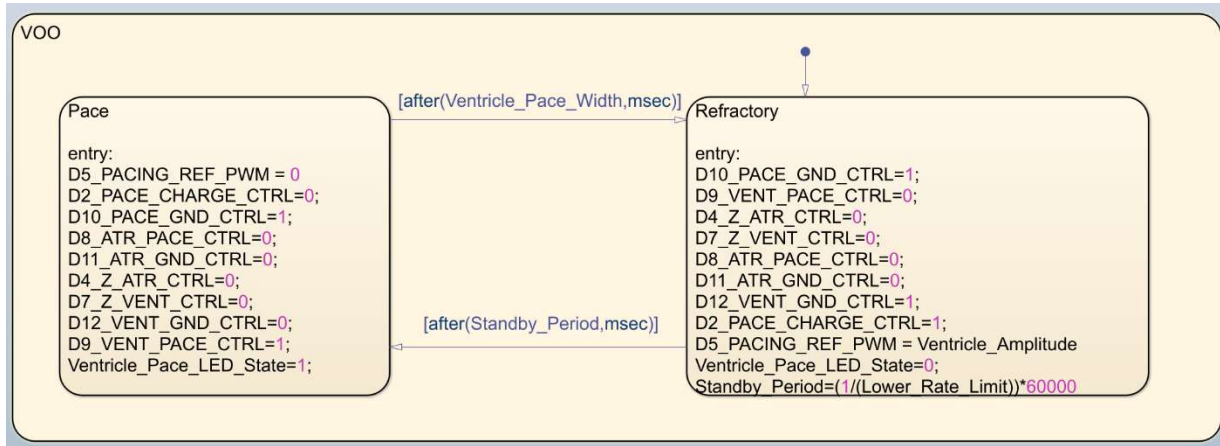


Figure 2 - VOO Stateflow

VOO is very similar to AOO described above, only swapping Atrium_Pace_Width to Ventricle_Pace_Width.

AAI

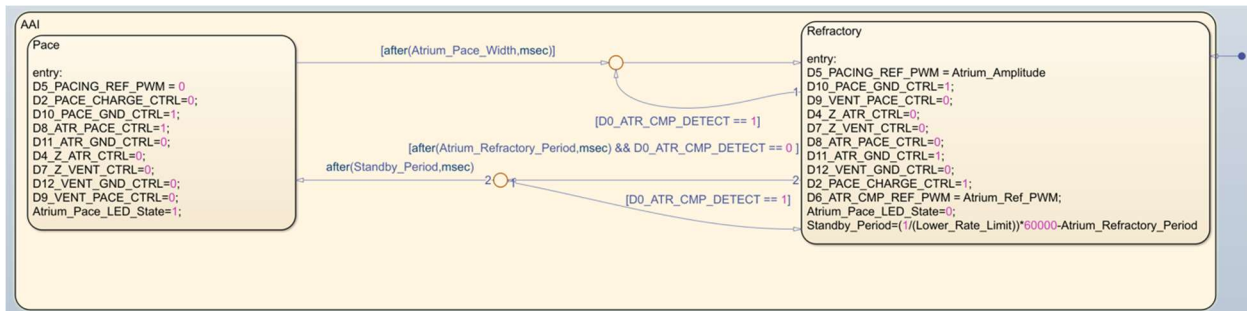


Figure 3 - AAI Stateflow

After all the output pins are set in Refractory, the state begins waiting for the refractory period to expire before moving onto the standby period. If it senses a natural pulse, it routes back to the beginning of the refractory state and resets the refractory period. If not, the state moves to the next junction where it begins waiting out the standby period. The standby period is calculated the same as AOO/VOO, but refractory period is subtracted from Standby to account for the time used up by the initial “after” statement. Like before, if it senses a natural pace, it returns to the beginning of Refractory and restarts the entire process, and if it does not sense a natural pace, the stateflow transitions to the Pace state. After all the output pins are set in Pace, the transition to refractory occurs after the Atrium_Pace_Width, which is a programmable parameter that defines the pulse width of the pace sent to the heart.

VVI

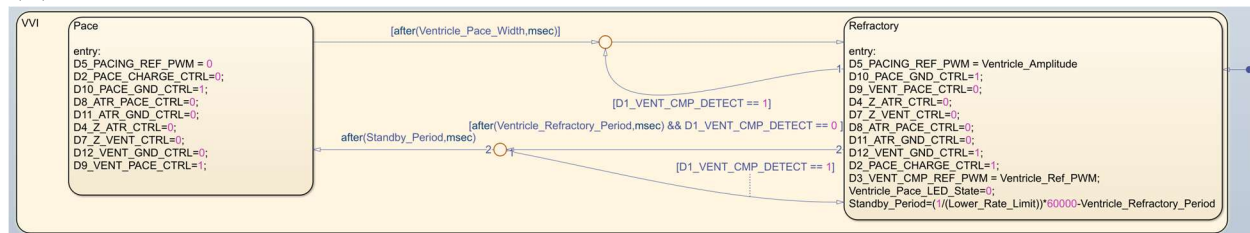


Figure 4 - VVI Stateflow

Like VOO to AOO, VVI is very similar to AAI, where the only difference is changing all the atrium variables to ventricle variables.

Mode Switching

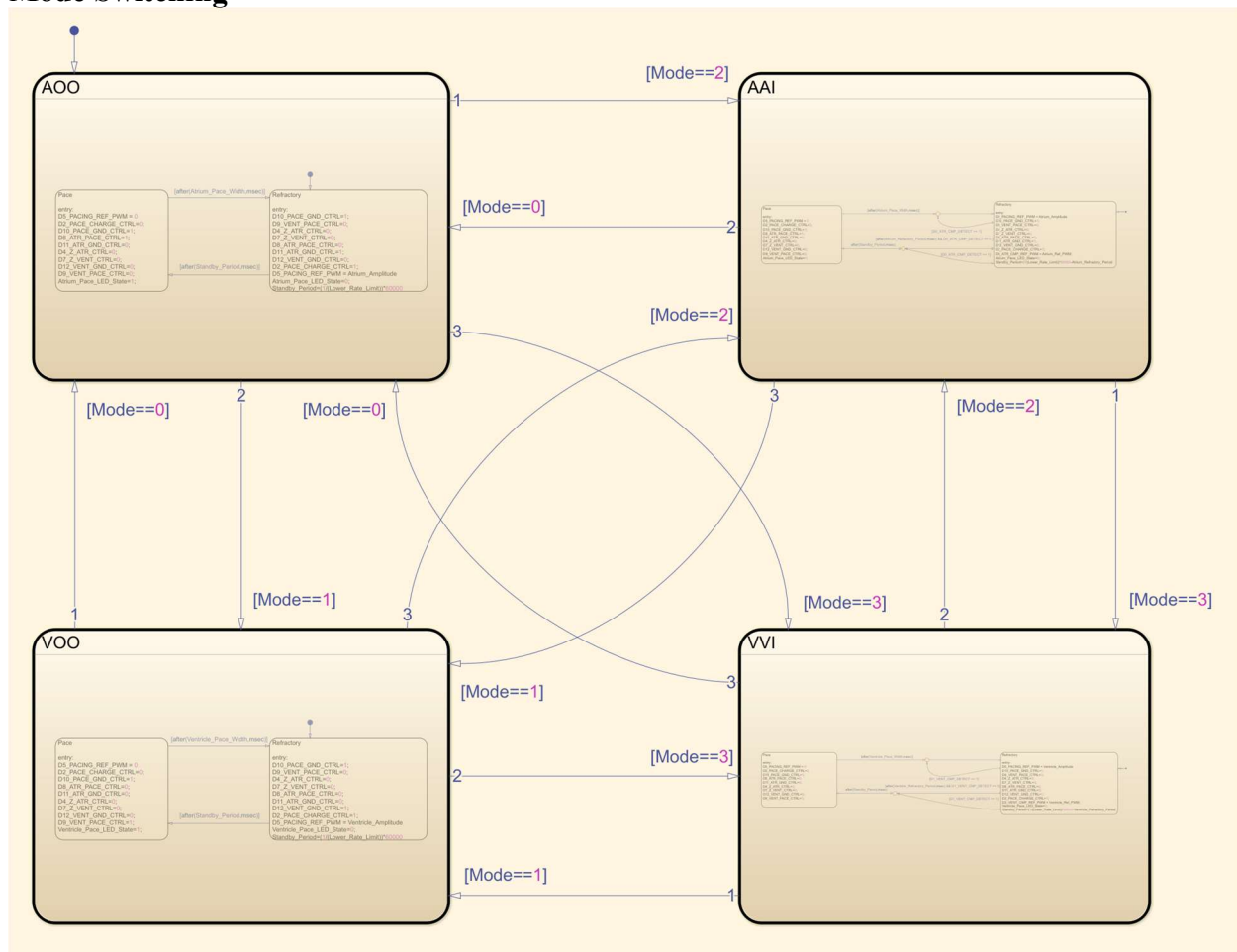


Figure 5 - Mode switching logic

Mode switching is accomplished by associating integer values to the various states as follows; AOO=0, VOO=1, AAI=2, VVI=3. Depending on the active state and the value of Mode, the stateflow follows the transition from the current state to the desired state.

Hardware Hiding

To ensure hardware hiding between the stateflow and the board and parameters, subsystems were created for hardware outputs and input parameters. Inside of the subsystems, inports were labelled and then routed to their corresponding pin out. In doing so the outputs/inputs of the stateflow could be routed through the inports so that they were not directly touching any digital outputs or inputs. On the input side logic blocks were used to hide any duty cycle conversions which are dependent on-board specific voltages.

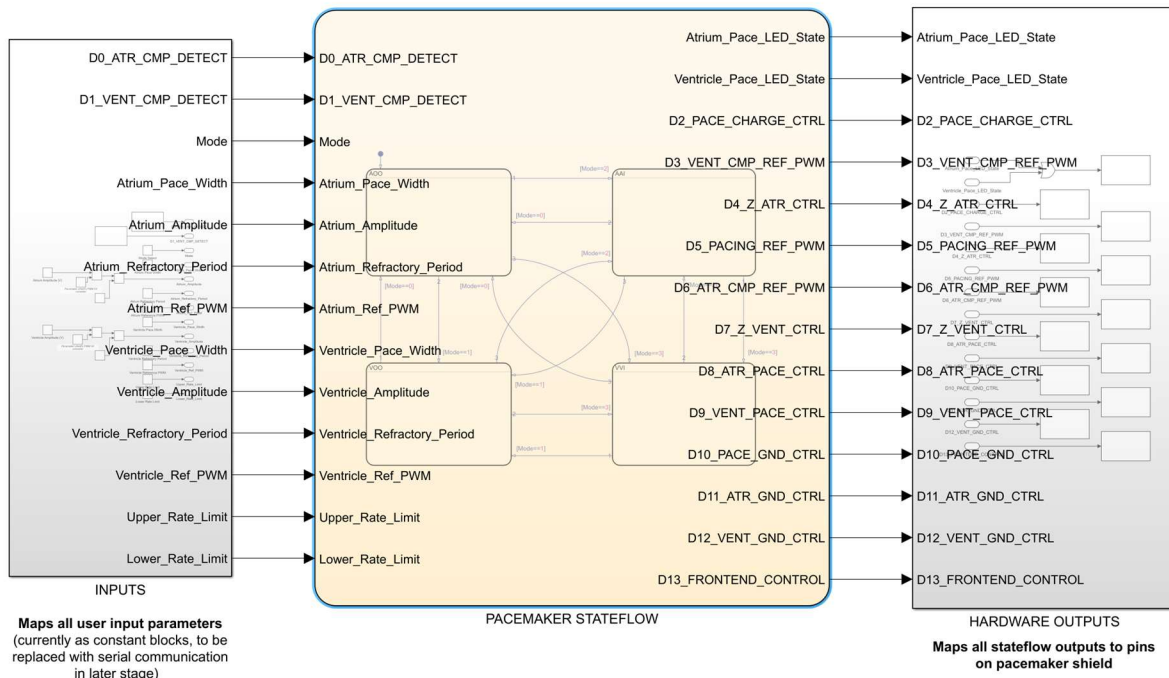


Figure 6 - Hardware hiding sub-systems

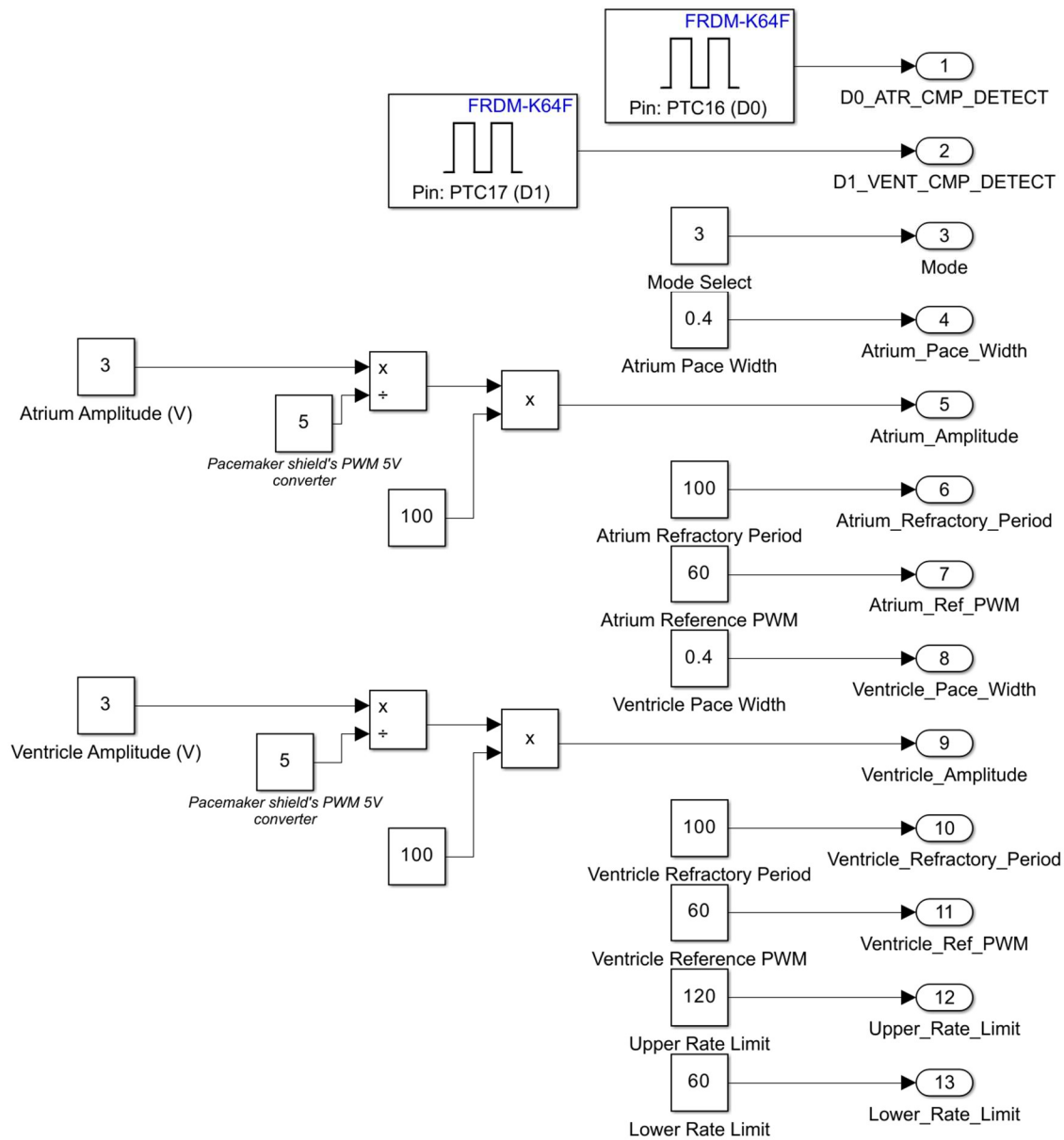


Figure 6 - Input Parameters

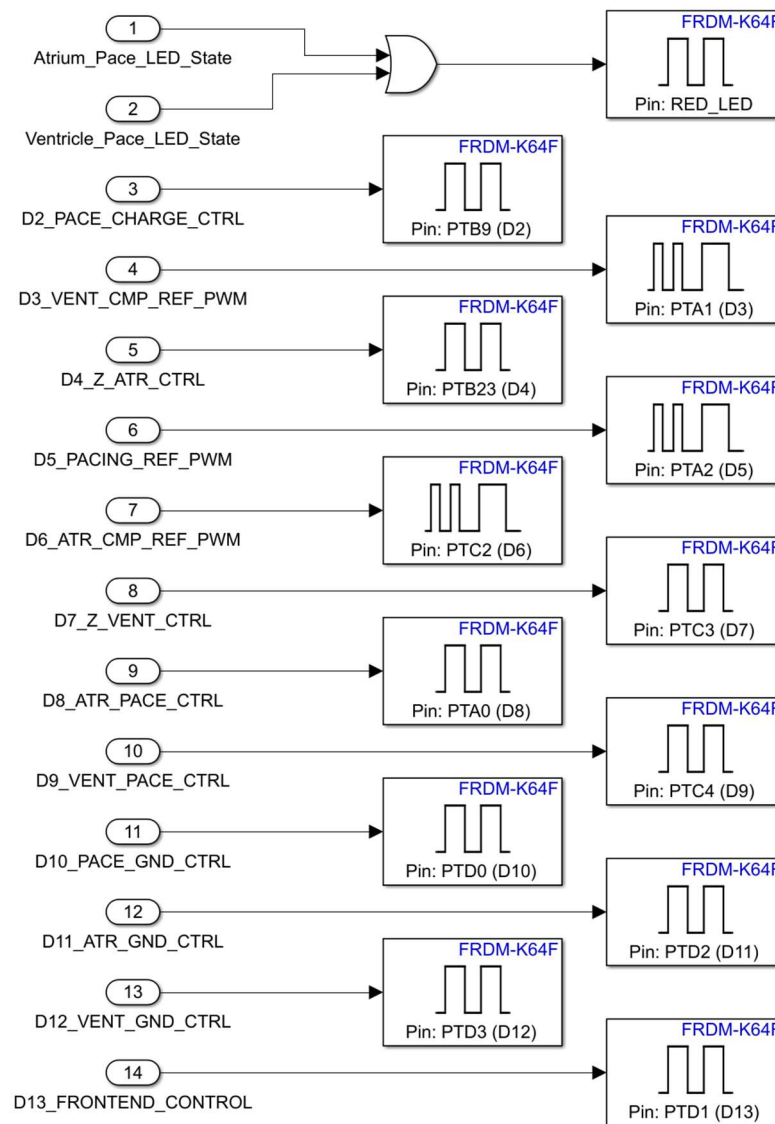


Figure 7 - Hardware Outputs

Challenges

One of the biggest challenges faced over the course of assignment one had to do with testing of stateflows and getting heartview to function properly. The majority of the initial testing was conducted on only one of the assigned boards, however it was failing to display any paced signals in any mode. It was later determined that communication between the pacemaker and heart on that particular board was faulty and that no signals were getting through. We were able to get our Simulink running on our other assigned board however it was only after quite a lot of wasted time working with a faulty board.

Another challenge faced was with sensing for AAI and VVI. Getting feedback from heartview was a struggle and after many hours of debugging it was determined a variable known

as Front end control was missed. In the circuitry this acts as the switch to enable the entire sensing circuitry and was the reason why no inputs were being received.

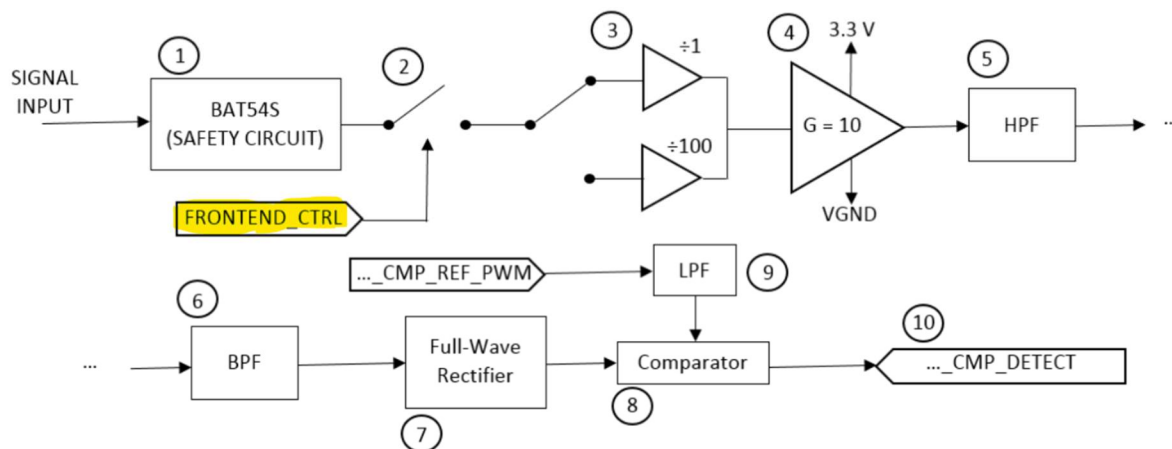


Figure 8 – Frontend Control

DCM Design

The development of the overall DCM design was separated into 4 distinct, interconnected categories that combined to create the DCM. Outside of input and output variables, each module is independent from each other and conducts their programs separately. Figure X shows our representation of our overall DCM design which involved 4 sections: login, registration, main, parameters.

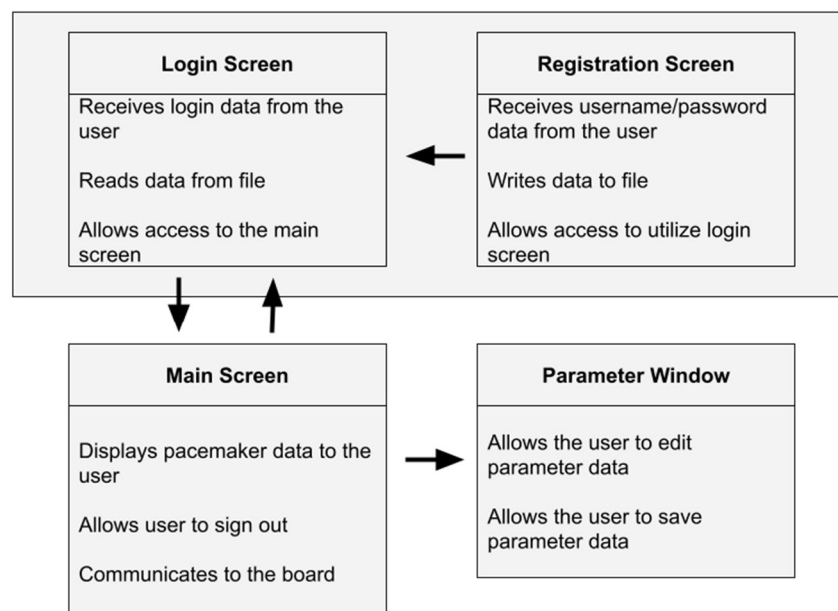


Figure 9 - Overall DCM design structure

Repository

To streamline our DCM design, we decided to create a GitHub repository that allows us easily make changes to our DCM prototype. Our overall structure aimed to have one main DCM

python file, with source and temp files around it to be imported into our final structure. Ideally, for assignment 2, we aim to keep graph data, and classes into separate header files to modularize the overall code structure further. To prevent data loss and git conflicts between different users we utilized separate GitHub branches to create certain features and merged them through pull requests to streamline our process.

Imported Libraries

Library Name	Description
import os	Text file reading and writing to store the accounts and the saved parameters
import customtkinter	Custom UI-based fork of python's tkinter library to create the pacemaker GUI
import hashlib	Encryption to secure user data in the textfile itself

Classes Descriptions

Class Name	Description
InputFrame()	Each individual parameter label, slider, value display and unit display
MessageWindow()	Pop-up window for alerting the user about important alerts regarding the login and registration screens
ParametersWindow()	Pop-up window allowing the user to insert customer parameter values that differ from the default
App()	The main display of the pacemaker application, where the user can print the report, run the program, select their mode, and sign out of the program

Functions

Each function is associated with the corresponding class name:

Function Name	Description
InputFrame() → <code>_init_()</code>	<p>A frame containing a slider and value label for input parameters.</p> <p>Args:</p> <ul style="list-style-type: none"> - master (Tkinter widget): The parent widget. - title (str): The title of the parameter. - from_ (float): The minimum value for the slider. - to (float): The maximum value for the slider. - ranges_and_increments (list or float): List of tuples specifying ranges and increments. - unit (str): The unit of the parameter.
update_slider_value()	<p>Update the displayed value when the slider is moved.</p> <p>Args:</p>

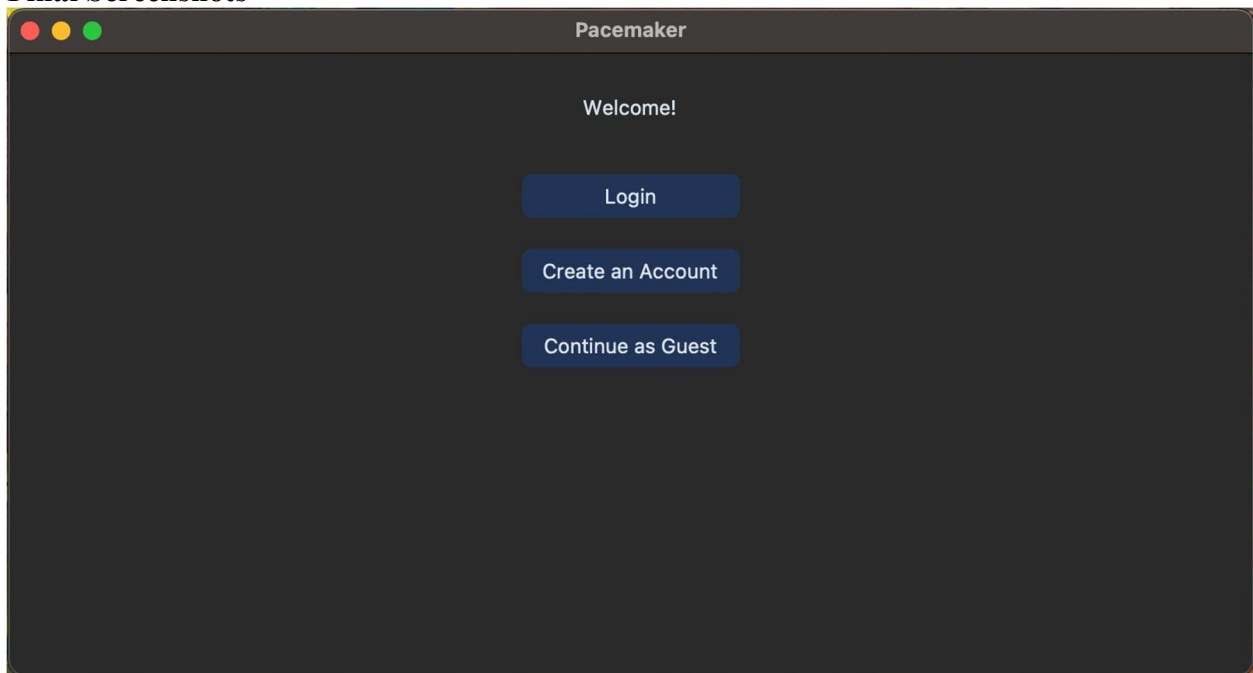
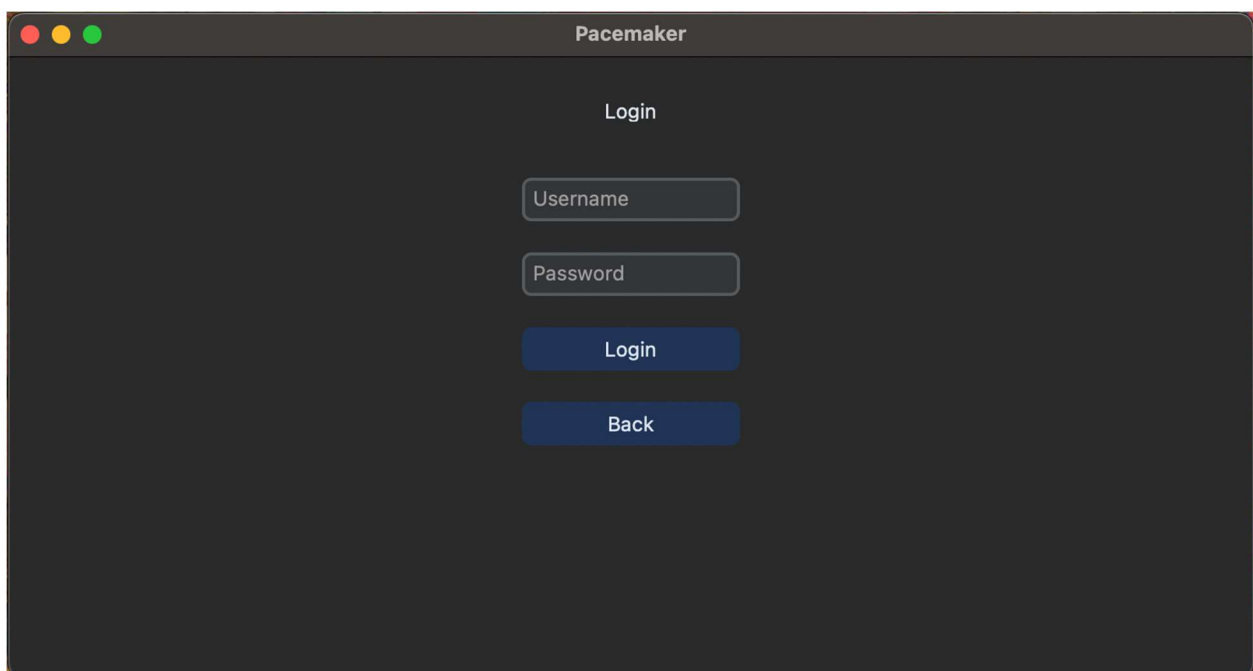
	- value (float): The current value of the slider.
variable_increment()	Adjust the value based on the specified ranges and increments. Args: - value (float): The current value. - rangeAndInc (list or float): List of tuples specifying ranges and increments. Returns: - float: The adjusted value.
process_ranges_and_increments()	Process the input ranges to ensure it is a list of tuples. Args: - ranges (list or float): List of tuples specifying ranges and increments. Returns: - list: Processed ranges
MessageWindow()→_init_()	A simple message window. Args: - title (str): The title of the window. - msg (str): The message to display.
ParametersWindow()→_init_()	A window for setting and saving parameter values. Args: - app (App): The main application instance.
get_user_list()	Retrieve the list of user accounts from the file. Returns: - list: List of user accounts.
reset_to_default()	Reset parameter values to default and save to file.
update_values()	Update the displayed values based on the stored values.
save_options()	Save the current parameter values to the file.
variable_increment()	Adjust the value based on the specified ranges and increments. Args: - value (float): The current value. - rangeAndInc (list or float): List of tuples specifying ranges and increments. Returns: - float: The adjusted value.
App()→_init_()	The main application class.
create_welcome_screen()	Create and display the welcome screen.

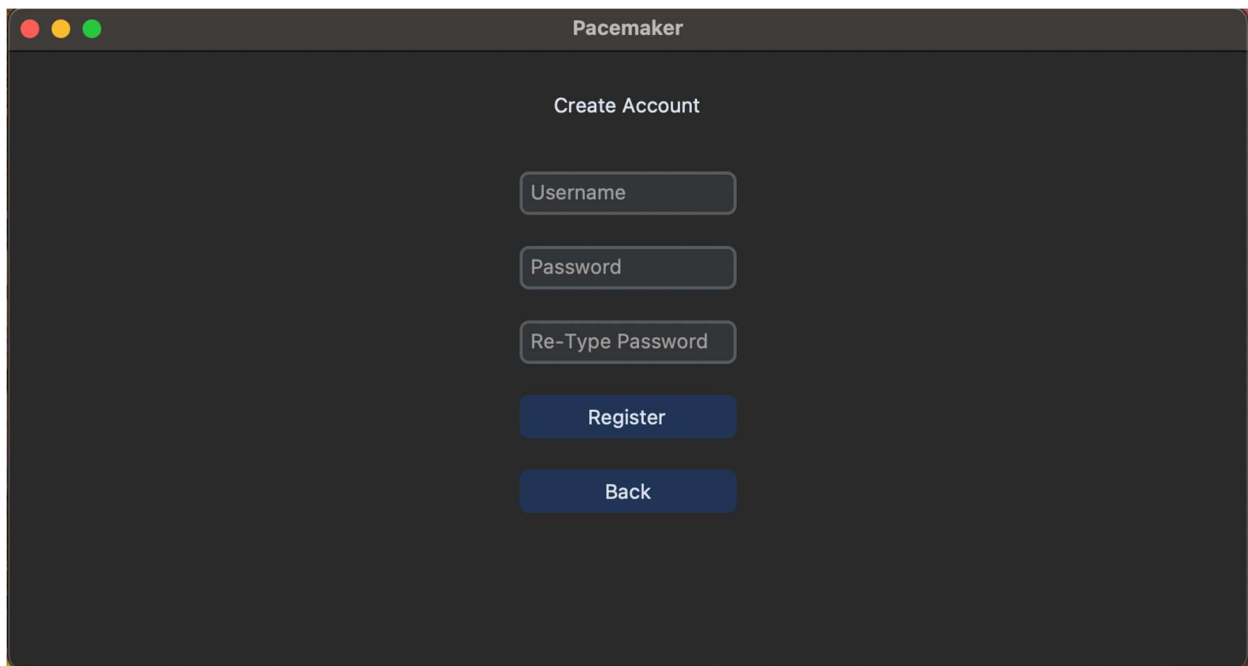
show_login_screen()	Switch to the login screen.
show_register_screen()	Switch to the register screen.
register()	Register a new user.
login()	Log in an existing user.
show_message()	Display a message window. Args: - title (str): The title of the message window. - msg (str): The message to be displayed.
get_user_list()	Retrieve the list of user accounts from the file. Returns: - list: List of user accounts.
username_exists()	Check if a username already exists. Args: - username (str): The username to check. Returns: - bool: True if the username exists, False otherwise.
back_to_welcome()	Return to the welcome screen.
optionmenu_callback()	Callback function for the option menu. Args: - choice (str): The selected option.
how_main_screen()	Switch to the main screen.
theme_event()	Toggle between dark and light themes.
show_parameters_popup()	Display the parameters window.

Problems and Challenges

Throughout the entirety of the design process for the DCM, we faced a variety of challenges that presented as follows:

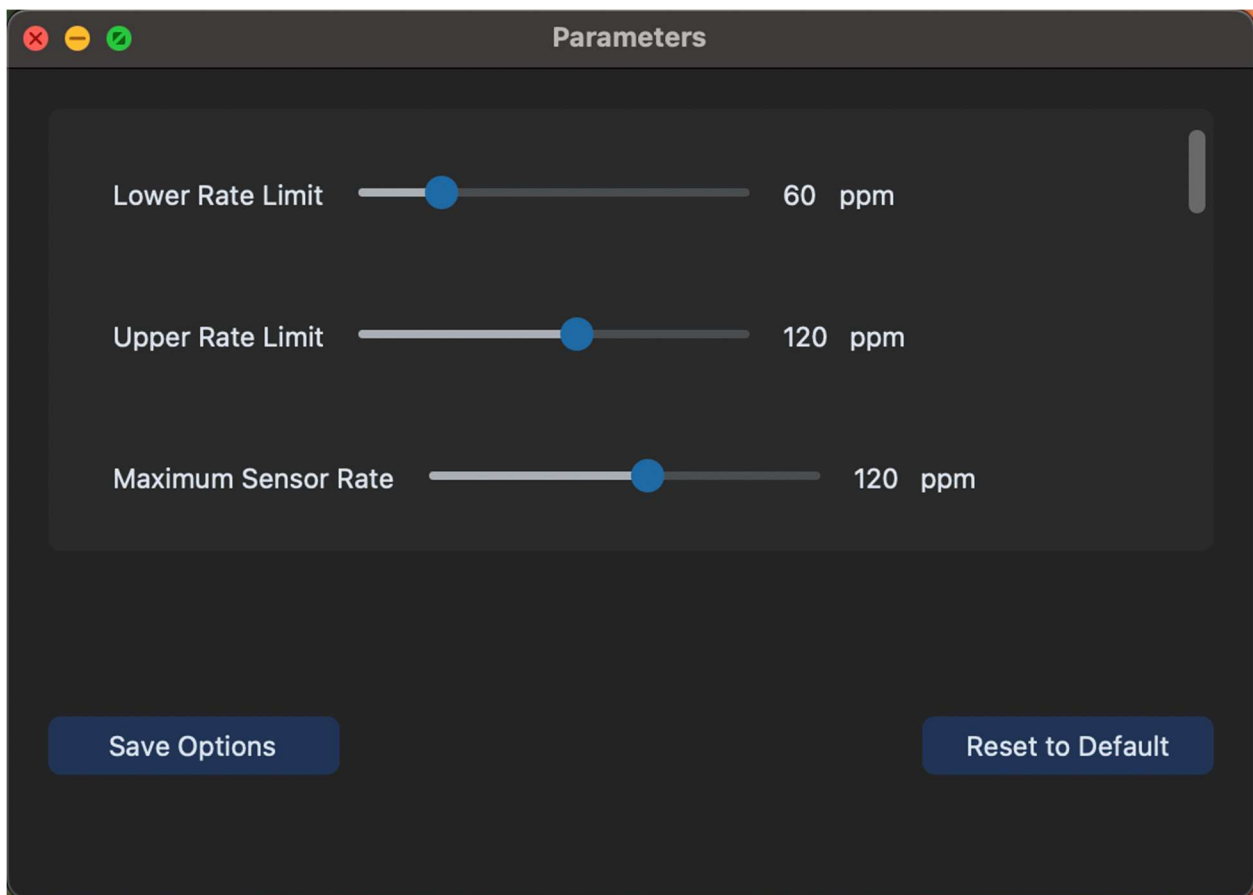
1. The first issue that we face was involving the creation of our parameter window. As we aimed to not hardcode each of the text, sliders, and slider captions with the end goal of creating cleaner code. The main issue that we had was accurately formatting and getting each of the elements to appear without any issues.
2. The second issue that we faced was that due to the separation of classes between the main and parameter window, we found a lot of issues inserting variables into the arguments of the class declarations. This was mainly due to errors involving customtkinter and prevented us from connecting the state variable from the App() class to the ParameterWindow() class.
3. Lastly, we found a lot of difficulties with our GitHub branches mainly due to merge conflicts created by poor programming practices

Final Screenshots*Figure 10 - DCM Home (Starting) Screen**Figure 11 - DCM Login Screen*



The image shows a macOS-style window titled "Pacemaker". Inside the window, the text "Create Account" is centered at the top. Below this text are three text input fields stacked vertically, labeled "Username", "Password", and "Re-Type Password". At the bottom of the form are two buttons: "Register" and "Back", both in a dark blue color.

Figure 12 - DCM Register Account Screen



The image shows a macOS-style window titled "Parameters". Inside the window, there are three sliders arranged vertically. Each slider has a label on the left, a blue knob in the middle, and a value with a unit on the right. The sliders are for "Lower Rate Limit" (set to 60 ppm), "Upper Rate Limit" (set to 120 ppm), and "Maximum Sensor Rate" (set to 120 ppm). At the bottom of the window are two buttons: "Save Options" and "Reset to Default", both in a dark blue color.

Figure 13 - DCM Parameter Modification Screen

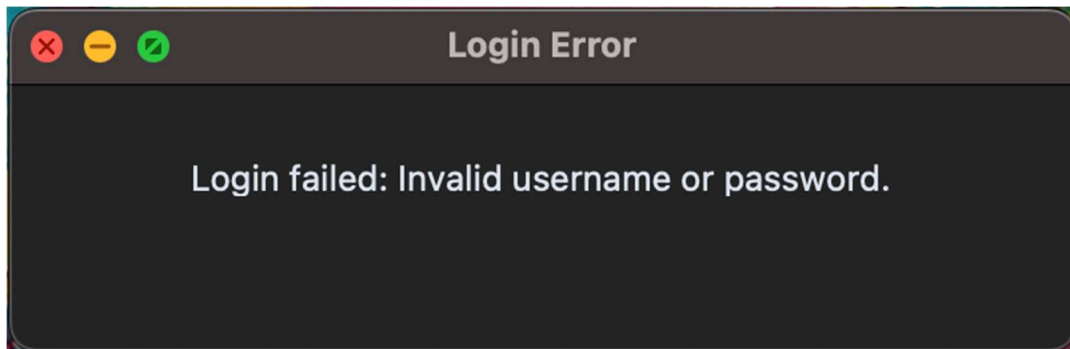


Figure 14 - DCM Sample Account Login Error Pop-Up

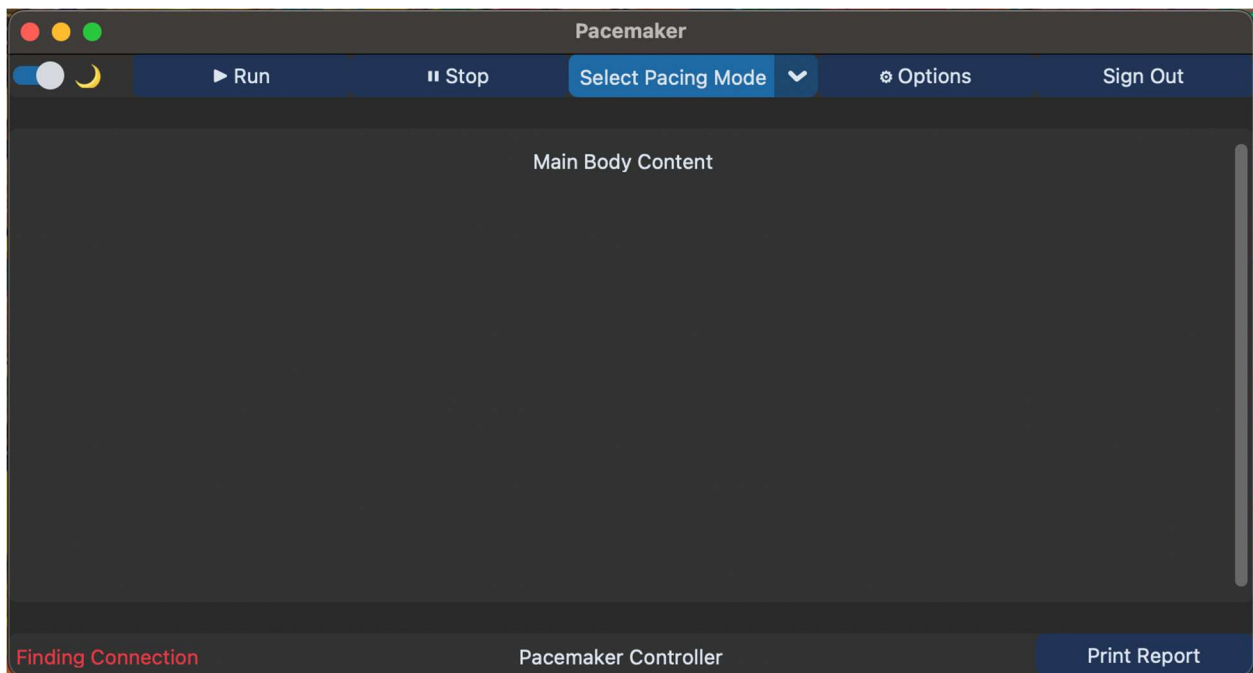


Figure 15 - DCM Main App Screen

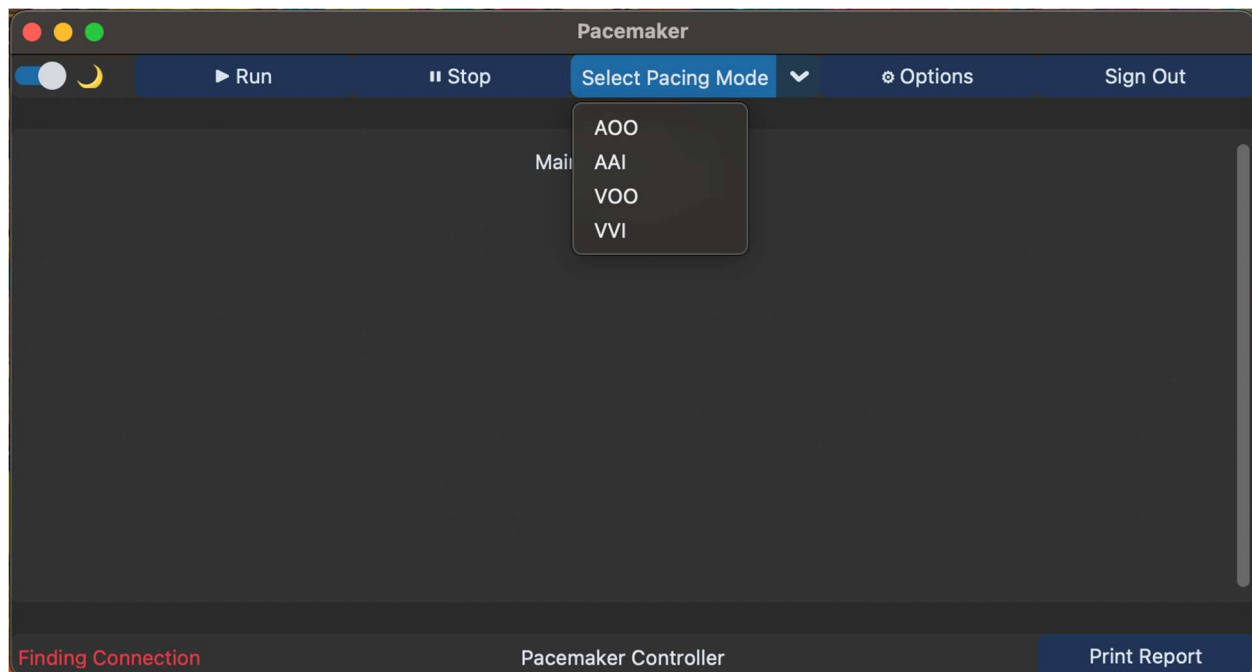


Figure 16 - DCM Main App Screen (Mode Selection)

Monitored Variables & Constants

AOO

The AOO

Pin name	Function
D5_PACING_REF_PWM	Charges Primary Capacitor
D2_PACE_CHARGE_CTRL	Starts and Stops capacitor charging (c22)
D8_ATR_PACE_CTRL	Discharges the primary capacitor through the atrium
D10_PACE_GND_CTRL	Controls the switch directly following the tip
D11_ATR_GND_CTRL	Used when discharging the blocking capacitor through the atrium to allow no charge buildup
D4_Z_ATR_CTRL	Used to analyze impedance of the atrial electrode
D7_Z_VENT_CTRL	Same as z atrial ctrl but for ventricle
D12_VENT_GND_CTRL	Same as atrial ground ctrl but for ventricle
D9_VENT_PACE_CTRL	Same as atrial pace ctrl but for ventricle
Atrium_Pace_LED_State	On when pacing
Standby_Period	Standby period between pace and refractory states
Atrium_amplitude	Amplitude of paced signal
Atrium_pace_width	During of pace signal

VOO

Pin name	Function
D5_PACING_REF_PWM	Charges Primary Capacitor
D2_PACE_CHARGE_CTRL	Starts and Stops capacitor charging (c22)

D8_ATR_PACE_CTRL	Discharges the primary capacitor through the atrium
D10_PACE_GND_CTRL	Controls the switch directly following the tip
D11_ATR_GND_CTRL	Used when discharging the blocking capacitor through the atrium to allow no charge buildup
D4_Z_ATR_CTRL	Used to analyze impedance of the atrial electrode
D7_Z_VENT_CTRL	Same as z atrial ctrl but for ventricle
D12_VENT_GND_CTRL	Same as atrial ground ctrl but for ventricle
D9_VENT_PACE_CTRL	Same as atrial pace ctrl but for ventricle
Ventricle Pace LED State	On when pacing
Standby Period	Standby period between pace and refractory states
Ventricle_amplitude	Amplitude of paced signal
Ventricle_pace_width	During of pace signal

AAI

Pin name	Function
D5_PACING_REF_PWM	Charges Primary Capacitor
D2_PACE_CHARGE_CTRL	Starts and Stops capacitor charging (c22)
D8_ATR_PACE_CTRL	Discharges the primary capacitor through the atrium
D10_PACE_GND_CTRL	Controls the switch directly following the tip
D11_ATR_GND_CTRL	Used when discharging the blocking capacitor through the atrium to allow no charge buildup
D4_Z_ATR_CTRL	Used to analyze impedance of the atrial electrode
D7_Z_VENT_CTRL	Same as z atrial ctrl but for ventricle
D12_VENT_GND_CTRL	Same as atrial ground ctrl but for ventricle
D9_VENT_PACE_CTRL	Same as atrial pace ctrl but for ventricle
Atrium Pace LED State	On when pacing
Standby Period	Standby period between pace and refractory states
Atrium_Refractory_Period	Duration where no paces can be delivered after pacing
D0_ATR_CMP_DETECT	Detects pace in atrium
D6_ATR_CMP_REF_PWM	Compares pwm signal for sensing
Atrium_amplitude	Amplitude of paced signal
Atrium_pace_width	During of pace signal

VVI

Pin name	Function
D5_PACING_REF_PWM	Charges Primary Capacitor
D2_PACE_CHARGE_CTRL	Starts and Stops capacitor charging (c22)
D8_ATR_PACE_CTRL	Discharges the primary capacitor through the atrium
D10_PACE_GND_CTRL	Controls the switch directly following the tip
D11_ATR_GND_CTRL	Used when discharging the blocking capacitor through the atrium to allow no charge buildup

D4_Z_ATR_CTRL	Used to analyze impedance of the atrial electrode
D7_Z_VENT_CTRL	Same as z atrial ctrl but for ventricle
D12_VENT_GND_CTRL	Same as atrial ground ctrl but for ventricle
D9_VENT_PACE_CTRL	Same as atrial pace ctrl but for ventricle
Venticle Pace LED State	On when pacing
Standby Period	Standby period between pace and refractory states
Ventricle_Refractory_Period	Duration where no paces can be delivered after pacing
D1_VENT_CMP_DETECT	Detects pace in atrium
D3_VENT_CMP_REF_PWM	Compares pwm signal for sensing
Ventricle_amplitude	Amplitude of paced signal
Ventricle_pace_width	During of pace signal

Decision Tables

Pacemaker

AOO

	Pace	Refractory
[after(Standby_Period,msec)]	1	0
[after(Atrium_Pace_Width,msec)]	0	1

VOO

	Pace	Refractory
[after(Standby_Period,msec)]	1	0
[after(Ventricle_Pace_Width,msec)]	0	1

AAI

	Pace	Refractory	Junction
[after(Atrium_Pace_Width,msec)]	0	1	*
[after(Atrium_Refractory_Period,msec)]	*	0	1
after(Standby_Period,msec)	1	*	0
[D0_ATR_CMP_DETECT == 1]	0	1	0

VVI

	Pace	Refractory	Junction
[after(Ventricle_Pace_Width,msec)]	0	1	*
[after(Ventricle_Refractory_Period,msec)]	*	0	1
after(Standby_Period,msec)	1	*	0
[D1_VENT_CMP_DETECT == 1]	0	1	0

DCM

Conditions (Inputs):					
Login Attempt	1	1	0	0	0
Account Creation	0	0	1	1	1
Matching Credentials in Database	1				
Max Users in Database	*	*	*	1	0

Actions (Outputs):					
Login Granted	1	0	0	0	1

<i>New Account Created</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>1</i>
-----------------------------------	----------	----------	----------	----------	----------	----------

<i>Conditions (Inputs):</i>								
VOO Mode	<i>1</i>	<i>1</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>
VVI Mode	<i>0</i>	<i>0</i>	<i>1</i>	<i>1</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>
AOO Mode	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>1</i>	<i>1</i>	<i>0</i>	<i>0</i>
AAI Mode	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>0</i>	<i>1</i>	<i>1</i>
VOO Parameter Changed	<i>1</i>	<i>0</i>	<i>1</i>	<i>0</i>	<i>1</i>	<i>0</i>	<i>1</i>	<i>0</i>
VVI Parameter Changed	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>
AOO Parameter Changed	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>
AAI Parameter Changed	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>	<i>*</i>

<i>Actions (Outputs):</i>								
<i>Change Allowed</i>	<i>1</i>	<i>0</i>	<i>1</i>	<i>0</i>	<i>1</i>	<i>0</i>	<i>1</i>	<i>0</i>

Testing

Testing the pacemaker state flow was completed by running each mode under several different heart conditions using Heartview. The signals that were used to conclude the state as functional are included below. The pacemaker shield's physical LED was also used for additional testing to ensure that the pacemaker was actively pacing.

All test conditions are run with nominal input parameters, 60bpm heart rate, and 30ms AV delay.

AOO & VOO Testing

As AOO/VOO is a write-only state, the monitored signals should show constant pulsing from the pacemaker to the atrium/ventricle (blue).

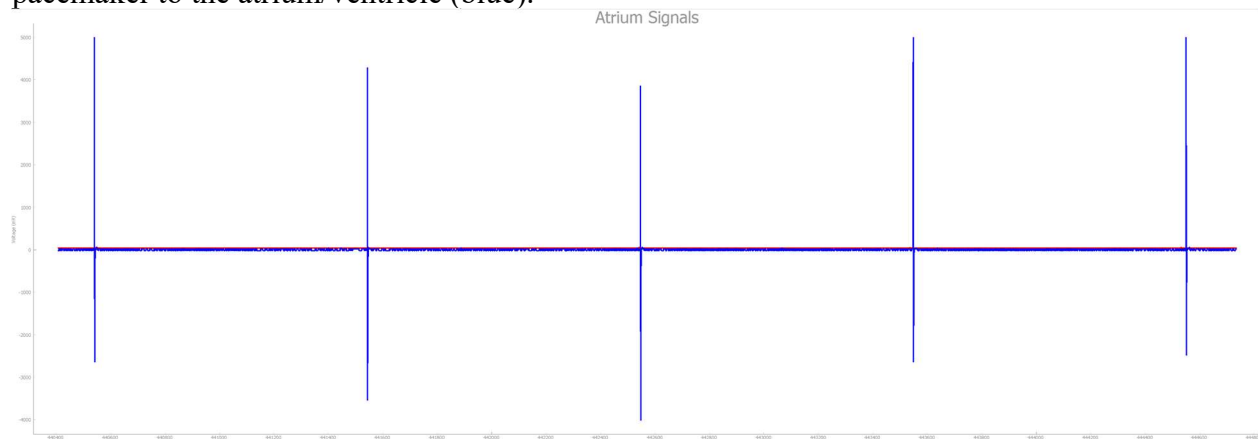


Figure 17 - AAO Test with natural atrium off

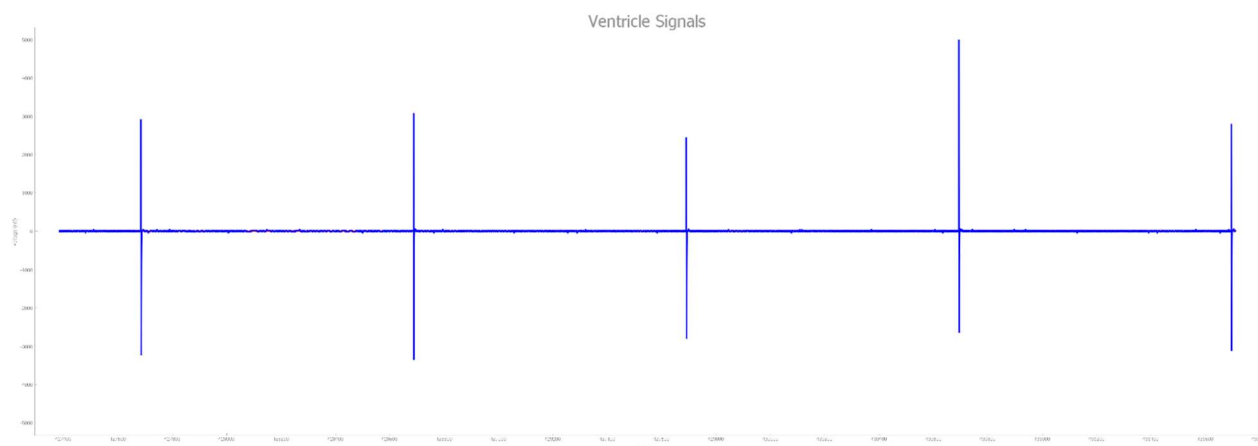
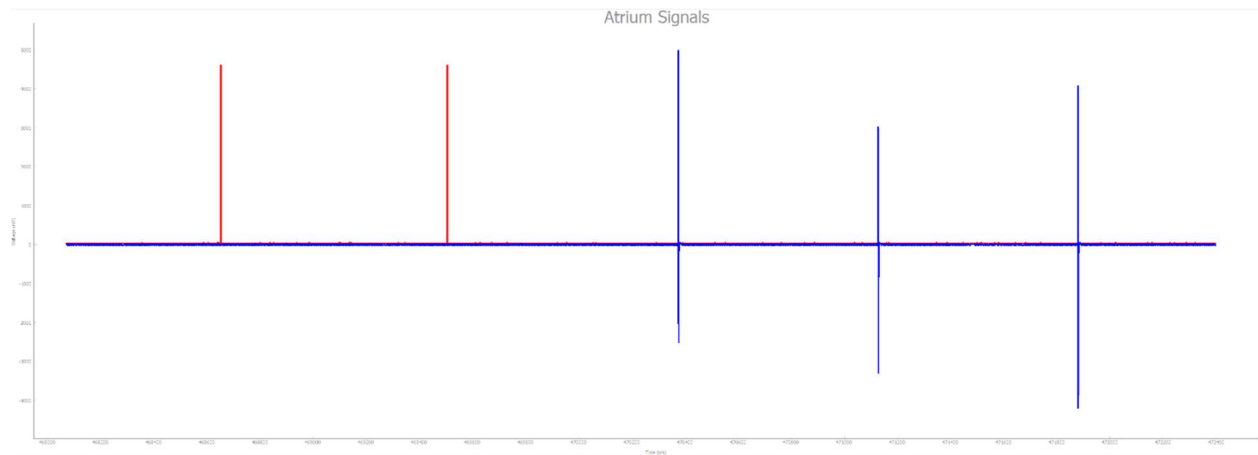
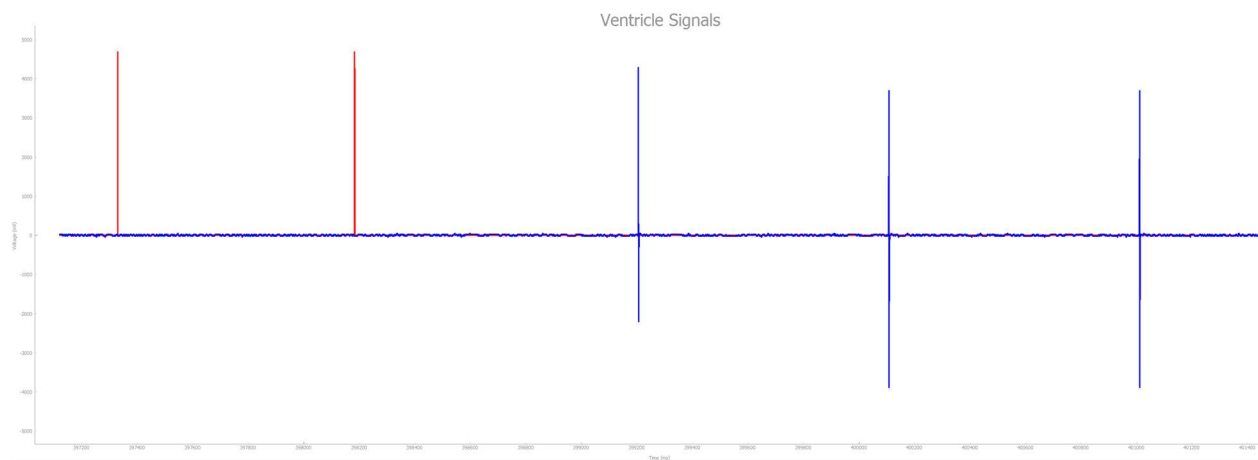
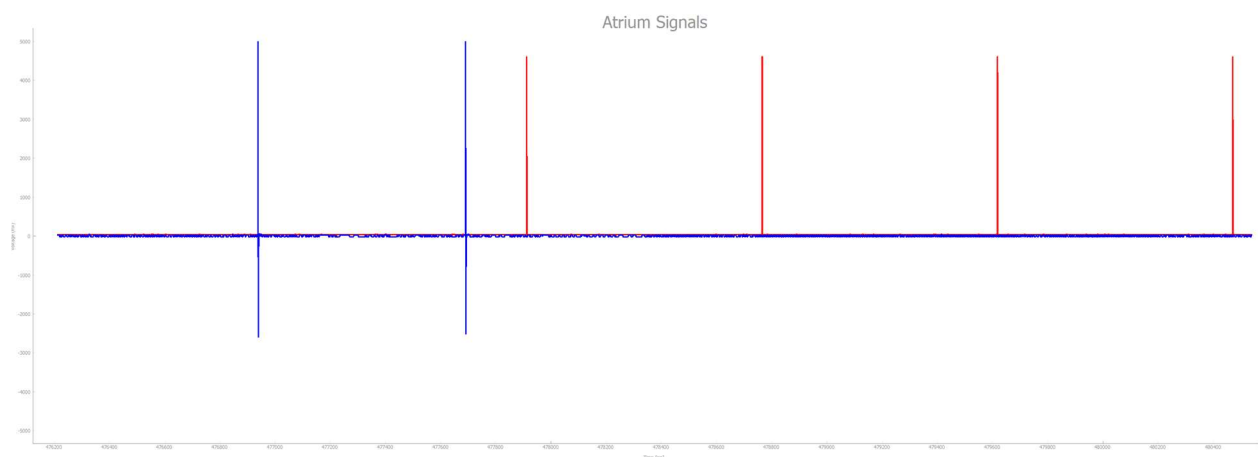


Figure 18 – VOO Test with natural ventricle off

AAI & VVI Testing

Functional signals for AAI/VVI show that when the natural atrium/ventricle is shut down (red), the pacemaker generates pulsing to takeover (blue). This is shown in figures [x](#) & [x](#). Additionally, the AAI/VVI modes should detect when the natural heart is beating again, and turn off pacing. This is shown in figures [x](#) & [x](#).

*Figure 19 - AAI natural to paced**Figure 20 - VVI natural to paced**Figure 21 - AAI paced to natural*

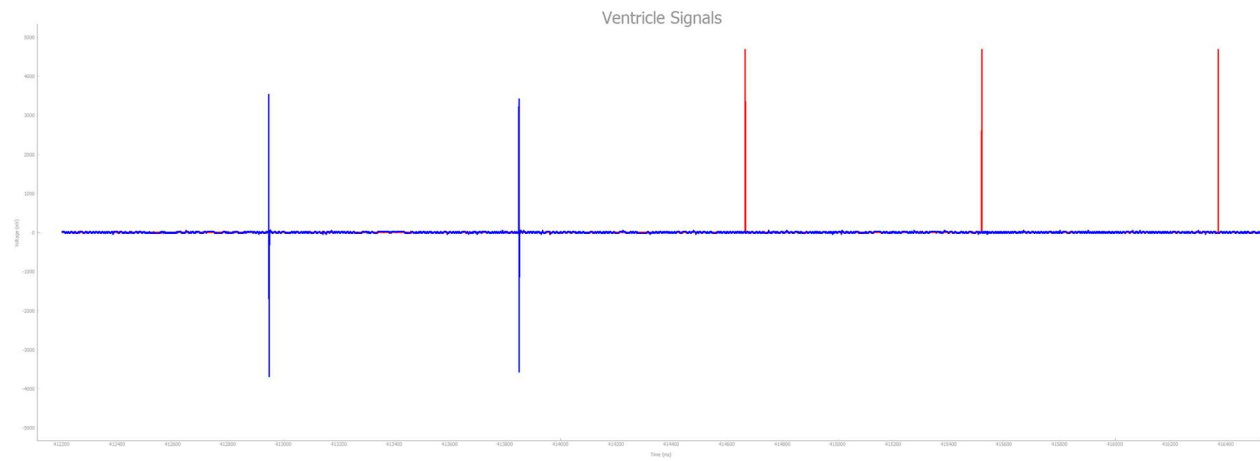


Figure 22 - VVI paced to natural