Assignment 2

MECHTRON 3K04: Software Development

Lucian Cheng - chengl26 - 400398983
Melody Yang - yangm98 - 400307007
Brian Manomaisupat - manomaib - 400383258
Ansh Mistry - mistra10 - 400182943
Rick Parikh - parikr7 - 400400451

Thomas Chiang

Lab L02

December 3, 2023

Table of Contents:

1.0 - Requirement Specifications **5**
1.1.1 - Pacemaker Requirements: REV 1 5
1.1.2 - DCM Requirements : REV 1 5
1.1.2.1 - Login Screen 6
1.1.2.2 - Registration Screen 6
1.1.2.3 - Main Interface (Client View) 6
1.1.2.4 - Main Interface (Admin/Physician View) 7
1.2.1 - Pacemaker Requirements : REV 2 7
1.2.1.1 - Programmable parameter subsystem 7
1.2.1.2 - Sensing subsystem 7
1.2.1.3 - Mode Stateflow chart 7
1.2.1.4 - Microcontroller input subsystem 7
1.2.2 - DCM Requirements : REV 2 8
1.2.2.1 - General Requirements 8
1.2.2.2 - Login Screen 8
1.2.2.3 - Registration Screen 8
1.2.2.4 - Main Interface (Client View) 8
1.2.2.5 - Main Interface (Admin/Physician View) 9
1.2.2.6 - About Screen 10
1.3.1 - Pacemaker Requirements: REV 3 (Current) 10
1.3.2 - DCM Requirements : REV 3 (Current) 11
1.3.2.1 - General Requirements 11
1.3.2.2 - Main Interface (Client View) 11
1.3.2.3 - Main Interface (Admin/Physician View) 11
1.3.3 - Safety Requirements : REV 3 (Current) 12
1.4.1 - Future/Anticipated Pacemaker Requirements 13
1.4.2 - Future/Anticipated DCM Requirements 14
1.4.2.2 - General Requirements 14
1.4.2.1 - Login Screen 14
1.4.2.2 - Registration Screen 14
1.4.2.3 - Main Interface (Client View) 14
1.4.2.4 - Main Interface (Admin/Physician View) 14
2.0 - Design Specifications **15**
2.1 - Pacemaker Design 15
2.1.1 - Programmable parameter subsystem 15
2.1.2 - Sensing subsystem 17
2.1.3 - Mode stateflow chart 17
2.1.4 - Microcontroller input subsystem 21
2.1.5 Serial Communication Implementation 22
2.1.6 Rate Adaptive Mode Implementation 25

2.2 - DCM Design 33
2.2.1 - General Design Choices 33
Theme 33
Language of Choice 33
Repository Structure 33
Imported Libraries 33
2.2.2 - Global Variables 34
Modes and Parameters 34
Colours 35
Functions 36
2.2.3 - Module Design 37
Main App Module 37
Methods 37
Variables 42
User Module 45
Methods 45
Variables 48
Successful Registration Prompt Module 48
Methods 48
Variables 49
Admin Login Module 49
Methods 49
Variables 50
Delete Account Module 50
Methods 51
Variables 51
Scrolling Programmable Parameters Frame Module 52
Methods 52
Variables 53
Electrogram Module 54
Methods 54
Variables 55
Serial Communication Module 55
Methods 55
Variables 56
2.2.4 - Final Application 57
2.2.4.1 - General Design Choices 57
2.2.4.2 - Login Screen Design Choices 58
2.2.4.3 - Registration Screen Design Choices 60
2.2.4.3 - Main Interface Design Choices 61
2.2.4.4 - About Screen Design Choices 71

2.3 - Future Pacemaker Design Changes 73
2.4 - Future DCM Design Changes 73
2.4.1 - General Future Design Changes 73
2.4.2 - Login Screen Future Design Changes 74
2.4.3 - Register Screen Future Design Changes 74
2.4.4 - Main Interface Future Design Changes 75
3.0 - Test Cases 76
3.1 - Pacemaker Test Cases 76
3.2 - DCM Test Cases 83
3.2.1 - Global Functions 83
Format_data(current_mode_data : dict): 83
Encrypt_password(password : str): 85
decrypt_password(encrypted_password : str): 85
3.2.2 - Main App Module Methods and Functionality 86
Pacemaker Connection 86
Pacemaker Disconnection 87
_start_button_cmd(self, selected_mode : str): 88
_stop_button_cmd(self): 88
3.2.3 - User Module 89
get_formatted_data(self): 89
3.2.4 - Scrolling Parameter Module 91
scroll_parameters_frame(master : customtkinter, current_mode_data : dict, current_mode : str, can_edit : bool, send_data_func : function, init_data_func : function, **kwargs): 91
3.2.5 - Serial Communication Module 91
list_serial_ports() 91
send_packet() 92
receive_packet() 92
get_egram_data() 92
4.0 - Assurance Case 93
5.0 - Development History 102
5.1 - Pacemaker Development 102
5.2 - DCM Development 108

1.0 - Requirement Specifications

1.1.1 - Pacemaker Requirements: REV 1

Implement stateflows using Simulink for the following pacemaker modes in permanent state: AOO, VOO, AAI, and VVI. The stateflow should use the following programmable parameters:

All the modes along with their corresponding parameters.

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

1.1.2 - DCM Requirements : REV 1

The following requirements listed are the preliminary requirements of the DCM at the beginning of its conceptualization. Many of these requirements may have been changed in lieu of the current design which is described in section 1.2.2 - Revised/Current DCM Requirements.

1.1.2.1 - Login Screen

- A user should have the ability to input a username and password to log into the main interface.
- A user should be notified if incorrect client credentials are entered and attempted.
- A user should be able to register a new account.
- A user should be able to change the language of the program.
- A user should be able to see the battery status.
- A user should be able to see the current date and time.
- A user should be able to see when the DCM and the Pacemaker are connected and communicating with one another.
- A user should see the amount of existing accounts stored, including the maximum number of accounts allowed to be stored.
- The maximum number of existing accounts should be limited to 10.
- A user should not be allowed to sign up for additional accounts once the account limit has been reached.

1.1.2.2 - Registration Screen

- A user should have the ability to input a username and password to sign up for an account.
- A user should be notified if a username is already taken by another account.
- A user should need to confirm their inputted password.

1.1.2.3 - Main Interface (Client View)

- A user should be able to log out of their account and return to the login screen.
- The user interface shall be capable of processing user positioning and input buttons.
- A user should be able to view four pacemaker modes (AOO, VOO, AAI and VVI) along with their corresponding parameters (PACEMAKER Document: Table 6)
- A user should select a pacemaker mode.
- A user should be able to only preview their corresponding stored parameters for each corresponding mode, including: (PACEMAKER Document: Table 7)
    - Lower Rate Limit
    - Upper Rate Limit
    - Atrial Amplitude
    - Atrial Pulse Width
    - Ventricular Amplitude
    - Ventricular Pulse Width
    - VRP
    - ARP
- A user should be defaulted to nominal values for each stored parameter.
- A user should be able to access an admin login page.
- A user should be able to see whether their current interface has admin permissions.

1.1.2.4 - Main Interface (Admin/Physician View)

- A user should only be able to access admin permissions after entering admin credentials in an admin login page.
- A user should be notified if incorrect admin credentials are entered and attempted.
- An admin should be able to log out of admin permissions.
- An admin should be able to delete a client account.
- An admin should be able to select a pacemaker mode.
- An admin should be able to adjust specific programmable parameters for each corresponding mode (AOO, VOO, AAI and VVI) including
    - Lower Rate Limit
    - Upper Rate Limit
    - Atrial Amplitude
    - Atrial Pulse Width
    - Ventricular Amplitude
    - Ventricular Pulse Width
    - VRP
    - ARP
- An admin should only be able to adjust programmable parameters within certain ranges and certain increments highlighted in PACEMAKER Document Table 7

1.2.1 - Pacemaker Requirements : REV 2

For the purpose of Assignment 1, the programmable parameters of PVARP, Hysteresis and Rate Smoothing are not required. The requirement of implementing mode AOO, VOO, AAI, and VVI remain. Additionally, hardware hiding must be applied to map the correct pins to the Simulink model.

1.2.1.1 - Programmable parameter subsystem

- Expandable to take values from DCM

1.2.1.2 - Sensing subsystem

- Reads ventricle and atrial pulses based on set voltage threshold.
- Outputs 1 when the signal is higher than the threshold voltage and 0 when the signal is lower than the threshold voltage.

1.2.1.3 - Mode Stateflow chart

- Takes programmable values from programmable parameter subsystem and sensing subsystem.
- Outputs values (High (1), Low (0), PWM (0-100) according to set pacing modes.

1.2.1.4 - Microcontroller input subsystem

- Takes input from Mode Stateflow chart and assigns to corresponding K94F microcontroller pins.

1.2.2 - DCM Requirements : REV 2

This section features revised and most up-to-date requirements with the finished product presented in Assignment 1 of the DCM portion. These requirements have been updated as the development and designing process of the DCM progressed and are better suited for the current view of the team in regard to the DCM portion.

1.2.2.1 - General Requirements

- A user on all screens should be able to view an “About” page of the application and navigate back.
- The user should be notified when the DCM and the Pacemaker are connected and communicating.
- The interface should only be written in English only.
- A user should be able to see the current date and time.
- A user should be able to see the battery status of the connected device, which should be displayed with values of N/A, BOL, ERN, ERT, and ERP.
- The window should not be resizable to avoid incorrect scale and layout configuration.

1.2.2.2 - Login Screen

- A user should be able to input a username and password to log into the main interface as a user.
- A user should be notified if incorrect client credentials are entered, attempted and redirected to the login screen again.
- A user should be able to register a new account.
- A user should be able to see the battery status.
- The maximum number of existing accounts should be limited to 10.
- A user should see the number of existing accounts stored, including the maximum number of accounts allowed to be stored, displayed as a fraction (X/10).
- A user should not be allowed to sign up for additional accounts once the account limit has been reached.
- When the correct credential combination of username/email and password is entered, and the “Sign In” button is clicked, the user should be logged in and be directed to the main interface screen.

1.2.2.3 - Registration Screen

- The register screen should allow the user to register for an account by inputting the following credentials: Email, Username, Password, and Password Confirmation
- The user should be prompted if the attempted username and/or email used to create a new account is already taken, and stay on the registration page.
- A password confirmation is required, and the user should be prompted if the two passwords do not match and stay on the registration page.
- In these three cases above, the user should not be able to register for an account until these issues are resolved.

1.2.2.4 - Main Interface (Client View)

- A user should be able to log out of their account and return to the login screen.
- The user interface shall be capable of processing user positioning and input buttons.

    - The interface should be able to determine where the user's cursor is, respond to appropriate button clicks, and detect any keyboard inputs.
- A user should be able to view 4 pacemaker modes (AOO, VOO, AAI and VVI) along with their corresponding parameters (PACEMAKER Documen)
- A user should be able to select a pacemaker mode.
- A user should be able only to review their corresponding stored parameters for each corresponding mode, including: (PACEMAKER Document: Table 7)
    - Lower Rate Limit
    - Upper Rate Limit
    - Atrial Amplitude
    - Atrial Pulse Width
    - Ventricular Amplitude
    - Ventricular Pulse Width
    - VRP
    - ARP
- A user should be defaulted to nominal values for each stored parameter.
- A user should be able to access an admin login page.
- The main interface should display the current permission status of the session as either client or admin privileges.
- The user should be able to download the bradycardia and temporary parameter reports.
- The current paced mode should be displayed to the user.
- A real-time electrogram graph should be displayed to the user.

1.2.2.5 - Main Interface (Admin/Physician View)

- A user should only be able to access admin permissions after entering admin credentials in an admin login page.
- A user should be notified if incorrect admin credentials are entered and attempted.
- An admin should be able to log out as admin to remove any admin privileges on this current session but should not sign out the currently logged-in user.
- An admin should be able to delete a user by inputting the correct admin credentials.
- An admin should be able to select a pacemaker mode.
- An admin should be able to **review and modify** specific programmable parameters for each corresponding mode (AOO, VOO, AAI and VVI) including
    - Lower Rate Limit
    - Upper Rate Limit
    - Atrial Amplitude
    - Atrial Pulse Width
    - Ventricular Amplitude
    - Ventricular Pulse Width
    - VRP
    - ARP
- An admin should only be able to adjust programmable parameters within certain ranges and certain increments highlighted in PACEMAKER Document Table 7
- The permission status shown to the user should be updated to admin privileges after correct admin password is entered.
- The user should be able to start and stop a current mode pacing at will based on the selected mode and its saved parameter data by clicking on the appropriate buttons ("Run" and "Stop").

1.2.2.6 - About Screen

- The application model number should be displayed to the user.
- The current software version of the DCM should be displayed.
- The serial number of the application should be displayed.
- The institution should be displayed.

1.3.1 - Pacemaker Requirements: REV 3 (Current)

A requirement for the pacemaker system is to implement functionality that increases the pulse rate based on activity detected by an accelerometer, essentially simulating what a normal heart would do in the duress of physical activity. The on-board accelerometer is to be used for this function and the rate at which the pulse rate operates is explained in the description for the rate adaptive state. For the implementation of stateflows using Simulink for the following pacemaker modes in permanent state: AOOR, VOOR, AAIR, and VVIR. These new modes are rate adapative versions of the previous set of 4 built previously. The goal of these modes is to have the ability to increase the pulse rate corresponding to activity detected based on accelerometer data as explained. The stateflows should use the following programmable parameters to facilitate the functionality of the system:

All 4 of the rate adaptive modes along with their corresponding programmable parameters.

Parameter	AOOR	VOOR	AAIR	VVIR
Lower Rate Limit	X	X	X	X
Upper Rate Limit	X	X	X	X
Maximum Sensor Rate	X	X	X	X
Fixed AV Delay				
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	
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

Additionaly, in order to enable the DCM to communicate with the simulink model and by extension the pacemaker board, a serial communication system is a required element to be implemented. The purpose is to allow full control of the programmable parameters highlighted above in the DCM application. This will require sending and receiving information from and to the pacemaker board.

1.3.2 - DCM Requirements : REV 3 (Current)

1.3.2.1 - General Requirements

- The DCM should be able to verify if parameters selected and parameters on the pacemaker are identical.
- The DCM should be able to verify if the pacemaker is connected at any point of time when it is running.

1.3.2.2 - Main Interface (Client View)

- A user should be able to view 4 additional pacemaker modes (AOOR, VOOR, AAIR, and VVIR) along with their corresponding parameters. (PACEMAKER Document: Table 6)
- A user should be able to open a real-time electrogram graph that is connected to their pacemaker.
- A user should be able to adjust parameters while having the real-time electrogram graph opened.
- A user should be able to interact with the main interface while the real-time electrogram graph is opened.
- A user should be defaulted to the last mode and corresponding parameters saved on their account when opening the DCM.
- A user should be able to log out while the real-time electrogram graph is opened

1.3.2.3 - Main Interface (Admin/Physician View)

- A user should be able to review and modify any new additions to the additional pacing modes including AOOR, VOOR, AAIR, and VVIR.
- A user should be able to modify any pacemaking mode while having the real-time electrogram graph opened.

- A user should be able to run or stop current modes while having the real-time electrogram graph opened.

1.3.3 - Safety Requirements : REV 3 (Current)

- The pacemaker and DCM shall identify and compare the stored pacing mode parameters from the DCM side to the stored values on the pacemaker to ensure they are identical to proceed with pacing. Identifying any errors in the two parameter sets will result in a shutdown of pacing to avoid improper pacing of the patient's heart. The user shall be notified when the parameters from the pacemaker and the DCM do not match and pacing shall halt.

  Rationale: It is unsafe if the paced mode and parameter set on the pacemaker do not perfectly match the parameters that the user inputted from the DCM side.

- The DCM shall attempt to avoid information breach failure to all saved users by implementing an encryption mechanism on saved user data (passwords)

  Rationale: Saved user data is important to keep private from external users and breach of information needs to be mitigated

- The pacemaker shall avoid any loss of pacing failure when connected and a user is logged in by resuming the previously saved last paced mode and its associated parameters of a logged in user and shall halt the pacing when logged out of a user

  Rationale: The pacemaker needs to help pace the heart of the patient and new information should not need to be re sent to initiate pacing

- The DCM and pacemaker shall be able to interpret whether it is supposed to read or write packets

  Rationale: Do not want to mis-using data or overwriting information needed for the patient

- The DCM and pacemaker shall avoid communication interception by third party software/hardware by implementing a safe and secure encryption technique for serial communication.

  Rationale: Do not want third party groups to be able to identify how data is sent/received between the pacemaker and DCM

- The DCM shall avoid parameter input and mode selection to be paced by unauthorized individuals through the implementation of an Admin system, designed to only be accessed by registered physicians and those who have competency in manipulating a pacemaker.

  Rationale: Do not want inexperienced individuals editing parameter data for pacing modes. Can cause improper pacing and puts the patients' lives at risk.

- The DCM shall protect the accounts of registered users and ensure they are able to log in when desired by implementing a “forgot password” feature to allow users to retrieve any forgotten credentials.

  Rationale: Do not want to lose accounts due to forgotten credentials. Can clog up the system (max 10 users) and may serve as a defect when patients cannot access their own medical records and conduct pacing

- The DCM shall attempt to avoid pacemaker issues involving its battery and life span by notifying the current active user about the connected pacemaker’s battery status to allow for future or immediate replacement

  Rationale: Do not want user or physician to be unaware of the battery status of the pacemaker, which could result in a loss of important information and data if the pacemaker automatically turns off due to lack of battery.
- The combined DCM and pacemaker device system shall avoid uncertain connection by displaying the connection status of the pacemaker on the DCM

  Rationale: A pacemaker should be properly connected and detectable by the DCM to ensure safe communication between the DCM and pacemaker

- The DCM shall avoid any dysfunctionality of the usability of the software and system when a pacemaker is not connected or has been disconnected willingly or unwillingly by the user. This ensures that the system can still be accessible and usable by the users to manipulate any patient data or retrieve reports. However many functions will be suppressed in the absence of a pacemaker to avoid traveling through unverified branches.

  Rationale: Do not want the GUI to crash or shut down just because the pacemaker is not plugged in since the physician should still be able to access and view the data that is present on the DCM. This can also avoid the loss of important information.

- The DCM and pacemaker shall avoid unrecognized parameter input failures through the implementation of fixed step values to represent each parameter to be interpretable easily between the pacemaker and DCM.

  Rationale: This will improve the safety of the pacemaker as the parameters will be of fixed intervals. This will also allow for the data transferred between the pacemaker and DCM to be safe and secure for the user, making it more reliable.

1.4.1 - Future/Anticipated Pacemaker Requirements

Considering that it is imperative for a software product to stay up to date and consistently improve functionality, it is crucial to highlight future requirements that may need to be implemented.

To ensure a complete product featuring all important functionalities of a heart, the remaining bradycardia operating modes shall be implemented into the pacemaker model along with the bradycardia states. The operating modes include; VDDR, DDIR, DOOR, DDD, VDD, DDI, DOO, VVT, AAT, OVO, OAO, ODO, and OOO. The remaining states are Permanent, Temporary, Pace-Now, Magnet, and Power-On Reset (POR). Adding functionality of these modes will ensure the pacemaker is capable of delivering thhe best and optimal performance.

1.4.2 - Future/Anticipated DCM Requirements

Future DCM requirements built off of the already established DCM requirements outlined in section 1.3.2.

1.4.2.2 - General Requirements

- The DCM should be able to be resized by any user.
- The DCM should give a warning when the battery of the pacemaker is below a certain percentage.
- The DCM should give a prompt when the pacemaker is at full battery.

1.4.2.1 - Login Screen

- The user should be able to request their password by submitting their associated email. If the email is in the system, then they shall be notified about their lost password.
- The user should be able to utilize two-factor authentication to log into the account.
- A user should be able to select the specific communication port, and be able to verify that it is an actual pacemaker device.

1.4.2.2 - Registration Screen

- The user should receive a confirmation email when successfully registering for an account with information about the DCM (serial number, institution, model number, version)
- The user should be prompted if the entered email address is an invalid email.

1.4.2.3 - Main Interface (Client View)

- A user should be able to close the real time electrogram graph using multiple methods.
- A user should be able to adjust the units of the real time electrogram graph
- A user should be able to resize the real time electrogram graph
- A user should be able to adjust the color scheme of the real time electrogram graph

1.4.2.4 - Main Interface (Admin/Physician View)

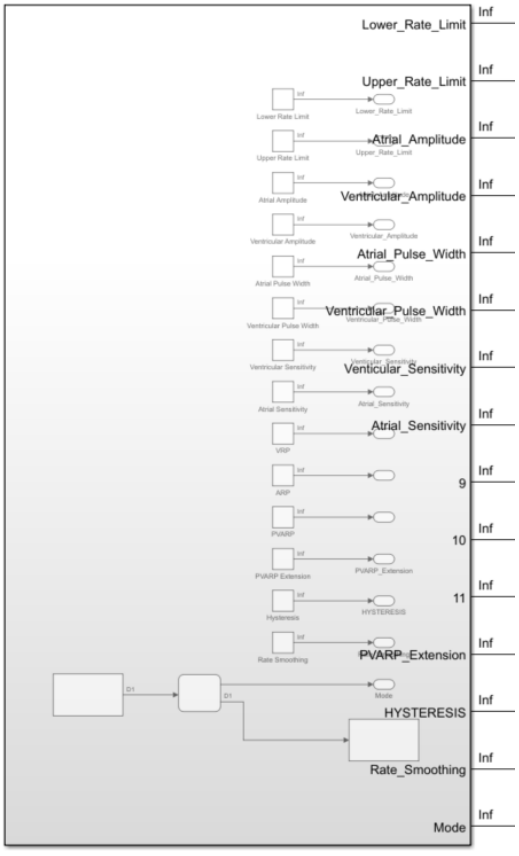
- When saving changes that were modified to the programmable parameters, the user should be prompted to enter the admin password once again to prevent any accidental pacing modifications, especially to a mode currently being paced. This will help to ensure safety so that parameters are not accidentally entered by someone other than the admin.
- A user should be able to open a bradycardia report that reflects the current electrogram graph data at the time the report was formed.

2.0 - Design Specifications

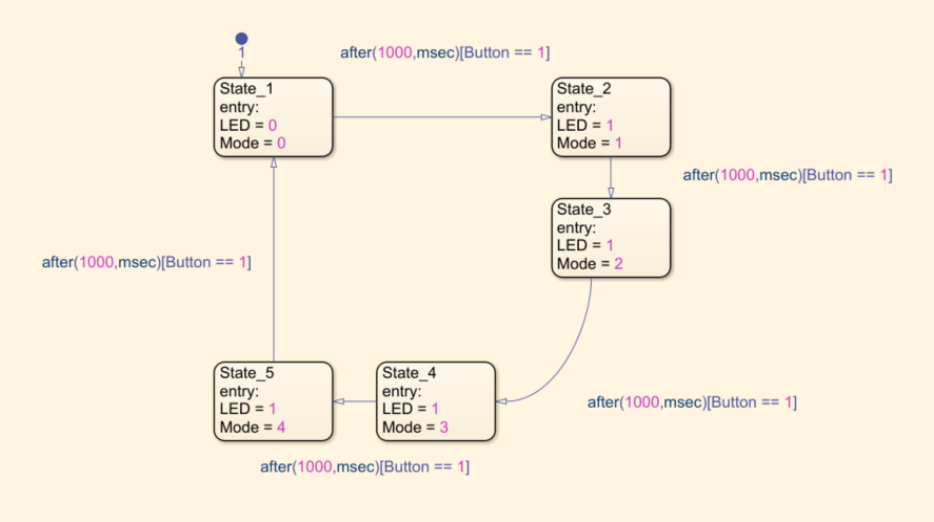
2.1 - Pacemaker Design

2.1.1 - Programmable parameter subsystem

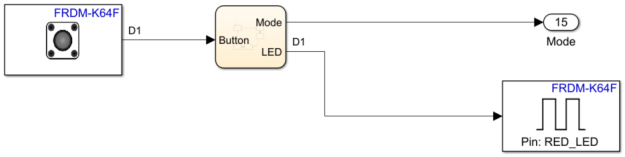
- Currently set to take integer values as inputs, set as nominal values for each parameter. In the future, these will be user-changeable using the DCM . Integer blocks were used for this purpose since they take less space than strings.
- Mode can be selected with the integer parameter “Mode.” “0” = OFF, “1” = AOO, “2” = VOO, “3” = AAI, “4” = VVI.
- A mode selection chart was added to change modes with the press of SW3. This functionality was added to make the demo process easier. LED is off when the mode is 0 or “OFF.” LED is on when the mode is 1,2,3,4 or AOO, VOO, AAI, VVI.



Programmable parameter subsystem.

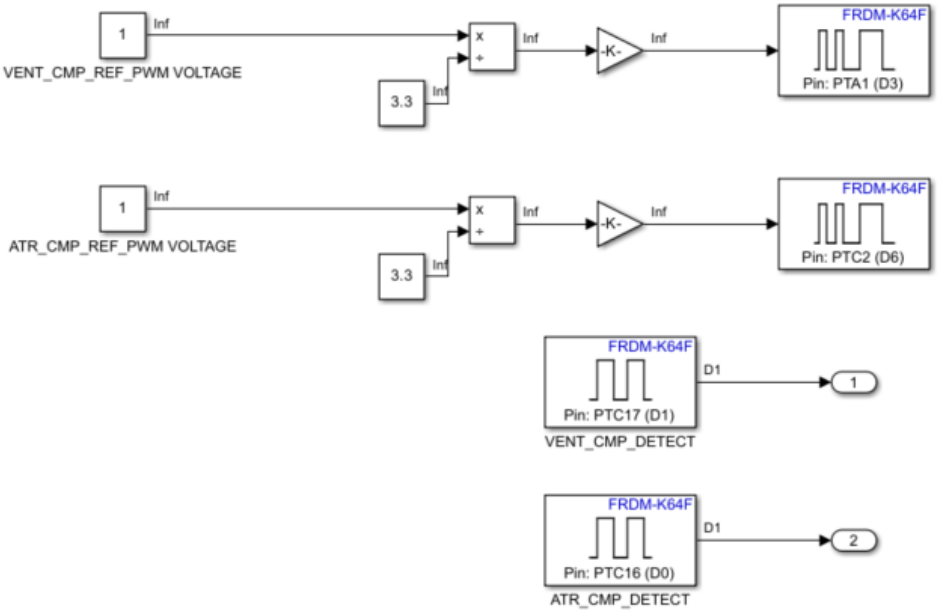


Mode selection chart



Mode selection system including the mode selection chart.

2.1.2 - Sensing subsystem

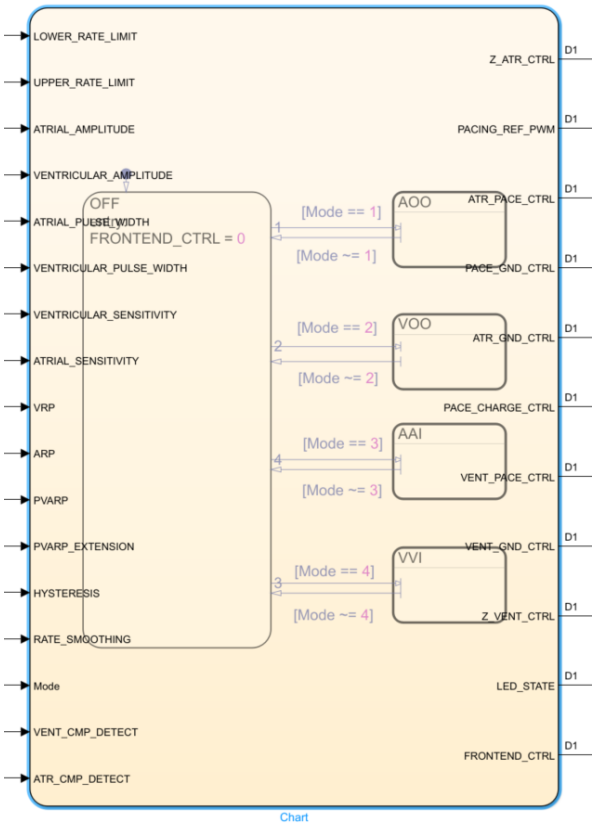


Sensing subsystem.

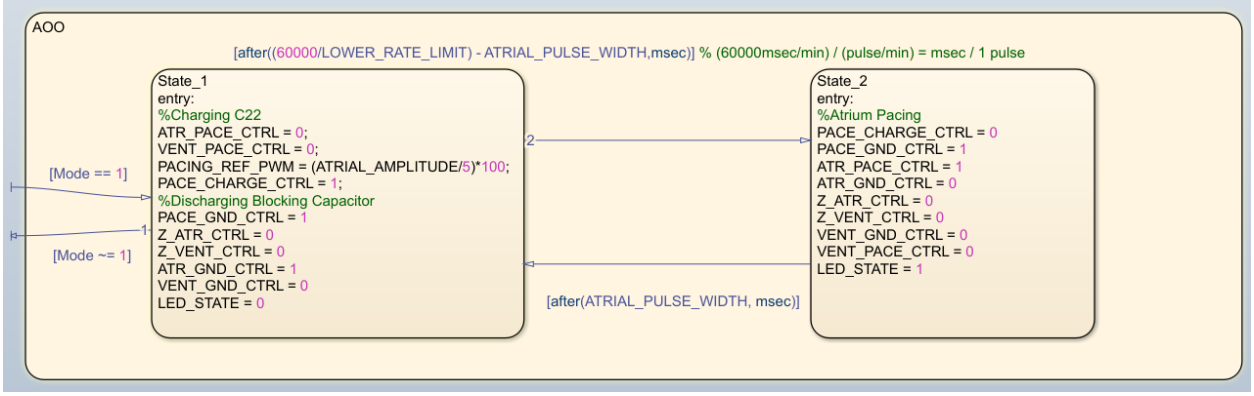
- The sensing subsystem is separate from the programmable parameter subsystem since the threshold voltage for sensing will not be user-changeable.
- In order to establish a threshold for when the ventricular or atrial action potential should be sensed, pin D3 (VENT_CMP_REF_ PWM) and pin D6 (ART_CMP_REF_PWM) uses PWM to charge a capacitor that will sustain a constant voltage for comparison.
- Since the capacitor voltage is linearly proportional to the duty cycle of PWM input, the input to pins D3 and D6 is saturated between 0 (0V) and 100 (3.3V). This is done by dividing the threshold voltage by 3.3V and multiplying by 100 to get a value between 0-100.
- Pin D1 (VENT_CMP_DETECT) and pin D0 (ATR_CMP_DETECT) are used as subsystem outputs for AAI and VVI modes.

2.1.3 - Mode stateflow chart

- The mode stateflow chart takes inputs from the programmable parameters subsystem and sensing subsystem. Depending on the operating mode of the pacemaker, it will output 1 (HIGH), 0 (LOW), and 0-100 (PWM) for input pins on the K94F microcontroller.

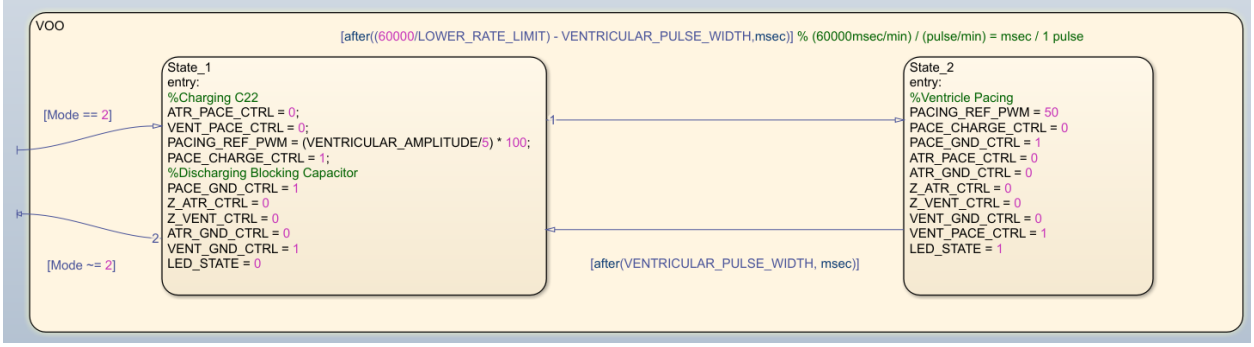


Mode stateflow chart



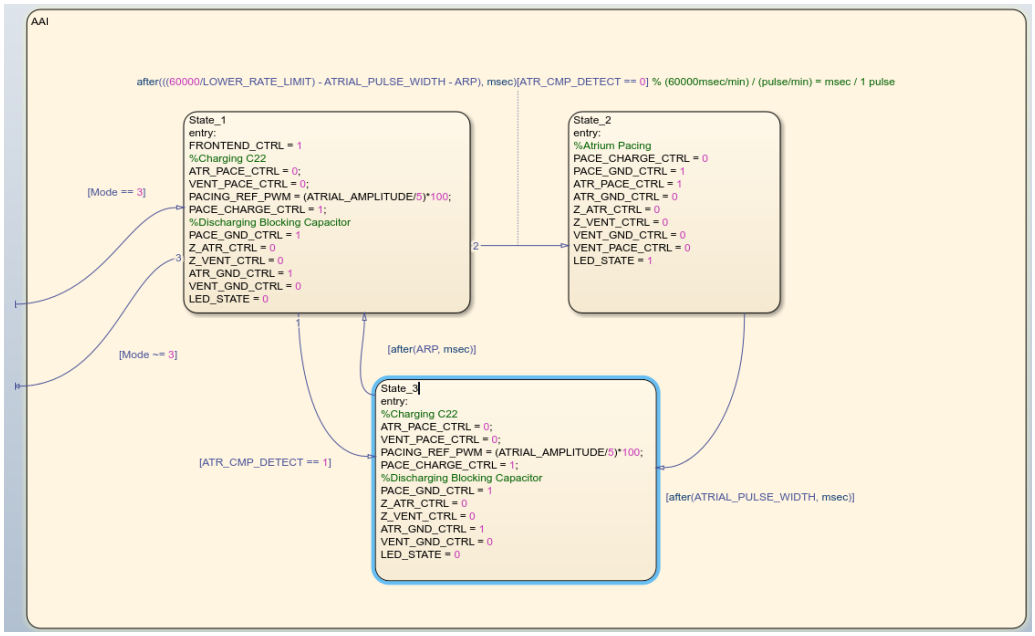
AOO mode

- AOO mode works by charging C22 using pin D5 (PACING_REF_PWM) to the input atrial amplitude. The voltage output by this pin saturates from 0V (0) to 5V (100). Therefore, the input ATRIAL_AMPLITUDE is divided by 5V and multiplied by 100 to result in a value between 0 and 100 to correspond to the atrial amplitude.
- State 1: ATR_PACE_CTRL, VENT_PACE_CTRL, PACING_REF_PWM, and PACE_CHARGE_CTRL are responsible for charging C22. The PWM signal ranges from 0 to 3.3 V, but the converter ranges the signal from 0 to 5 V. PACE_CHARGE_CTRL is set to 1 because we need to use PACING_REF_PWM to charge primary capacitor C22.
- State 1: PACE_GND_CTRL, VENT_PACE_CTRL, Z_ATR_CTRL, Z_VENT_CTRL, ATR_PACE_CTRL, ATR_GND_CTRL and VENT_GND_CTRL are responsible for discharging blocking capacitor. This is required to reduce the net charge build-up to zero in the atrium. PACE_GND_CTRL is also set to 1 to allow current to flow from the ring to the tip in the atrium. ATR_GND_CTRL is set to 1 to allow the discharging of the blocking capacitor through the atrium to allow no charge buildup.
- State 2: PACE_CHARGE_CTRL, PACE_GND_CTRL, ATR_PACE_CTRL, ATR_GND_CTRL, Z_ATR_CTRL, Z_VENT_CTRL, VENT_GND_CTRL, VENT_PACE_CTRL are responsible to atrial pacing. Since ATR_PACE_CTRL is set to 1, the charge will flow through the switch and accumulate in the blocking capacitor (C21).



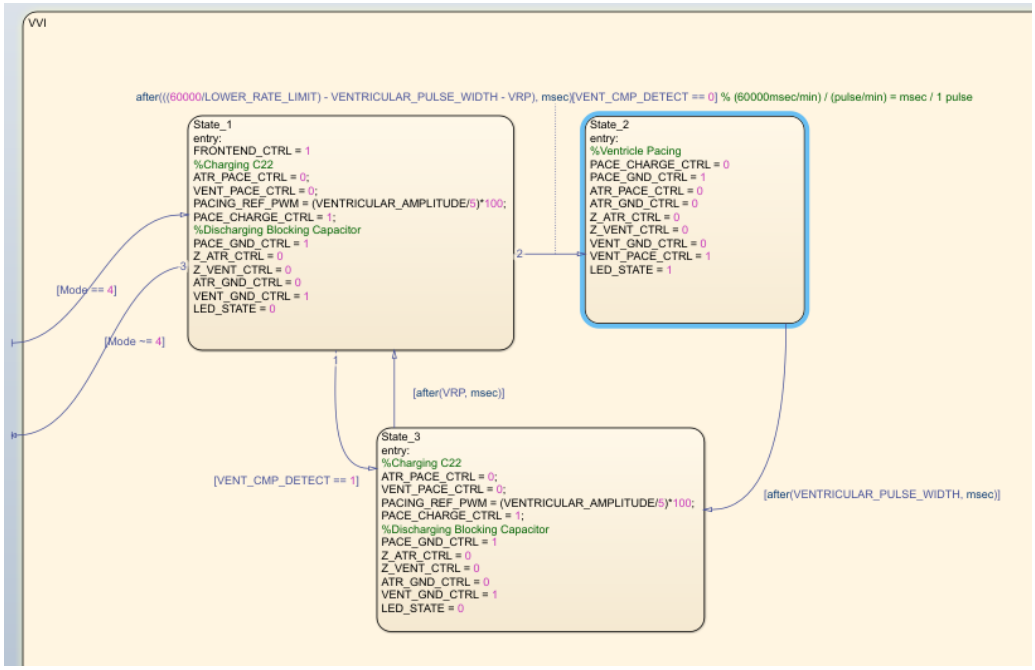
VOO mode

- VOO mode works similarly to AOO mode. The difference is in the pins being activated.
- State 1: VENT_GND_CTRL is set to 1 instead of ATR_GND_CTRL.
- State 2: VENT_PACE_CTRL is set to 1 instead of ATR_PACE_CTRL.



AAI mode

- State 1 and State 2 of AAI mode are identical to AOO. There is an addition of State 3, which is a state in which the pacemaker waits after an atrial event for the time duration of the programmable value ARP. In this state, atrial events shall not inhibit nor trigger pacing.
- The variable FRONTEND_CTRL enables the sensing circuit in the pacemaker for the sensing subsystem to work. The ATR_CMP_DETECT value from the sensing subsystem is then used to detect an atrial event to move from State 1 to State 3.

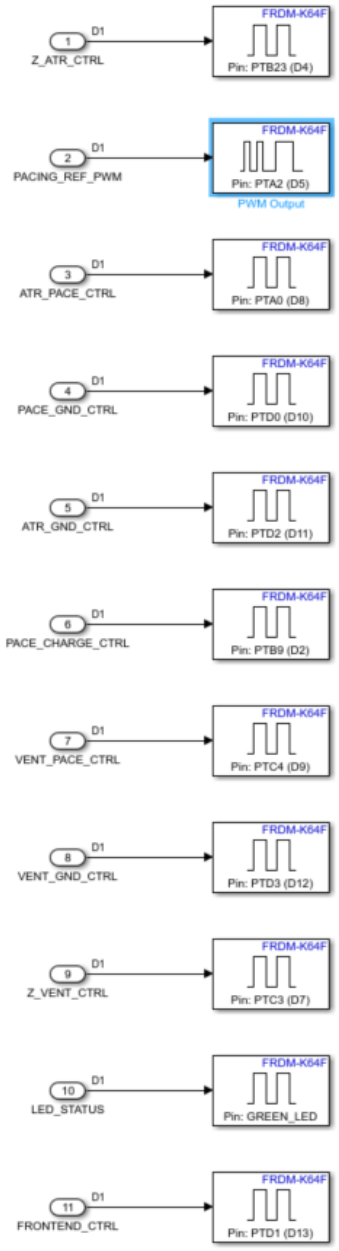


VVI mode

- VVI mode is similar to AAI mode, with the difference being the pins set to 1. In State 1 and State 3, VENT_GND_CTRL is set to 1 instead of ATR_GND_CTRL. In State 2, VENT_PACE_CTRL

is set to 1 instead of ATR_PACE_CTRL. Ventricle pulse is detected, and VRP and VENTRICULAR_PULSE_WIDTH are used.

2.1.4 - Microcontroller input subsystem

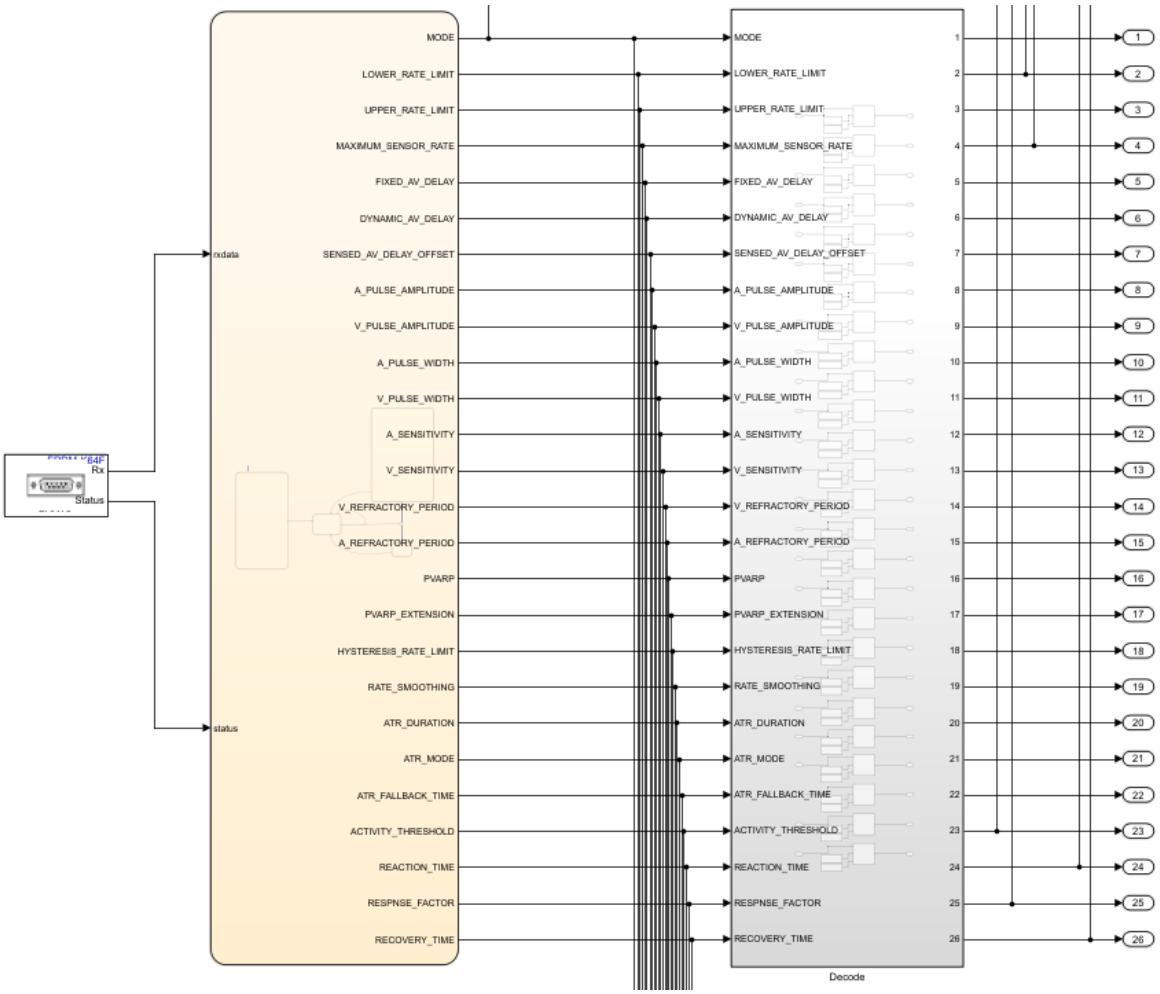


Microcontroller input subsystem

- This subsystem takes the output from the Mode Stateflow chart, allocated to pins on the K94F microcontroller.

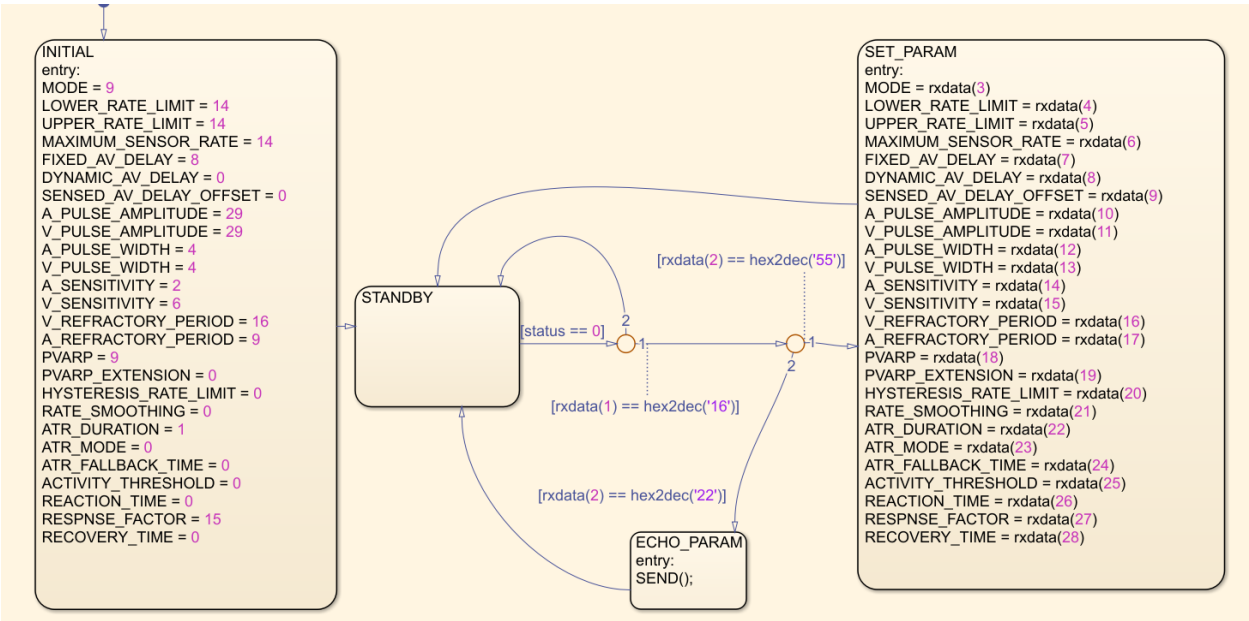
2.1.5 Serial Communication Implementation

With the additional requirements for the rate adaptive modes and the serial communication, they were carefully assessed and implemented with caution to provide a seamless function for the user of the pacemaker and DCM. With the implementation, the programmable parameters can be set in totality through the DCM. This creates the seamless design of the system that was intended from the beginning. The same principles still apply for how to change between the modes for instance.



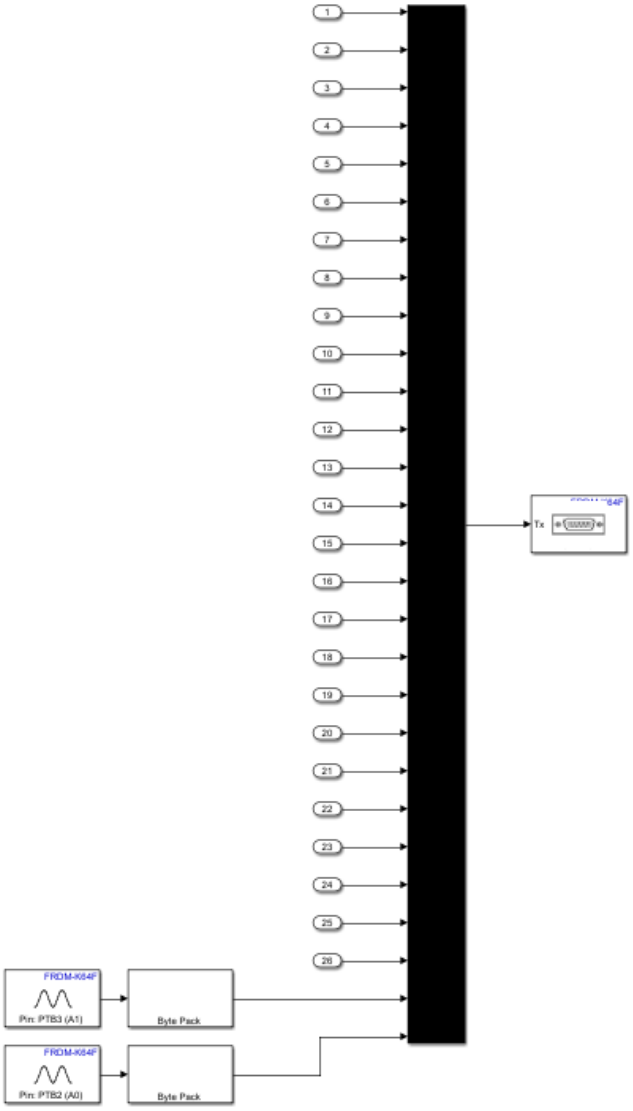
Serial communication main work flow

To carry forward the same hardware hiding principles, 1 subsystem was created to implement the serial communication feature on the simulink model. There are 3 main parts for serial communication. Please refer to the figure below. First, receiving data from the serial port on the board which is located in the figure on the left. This serial port receives 28 bytes of data which is the stored useful information containing the data of the programmable parameters. When the first byte is 16 and the second byte is set to 55, they represent the sync bytes for receiving a signal from the DCM.



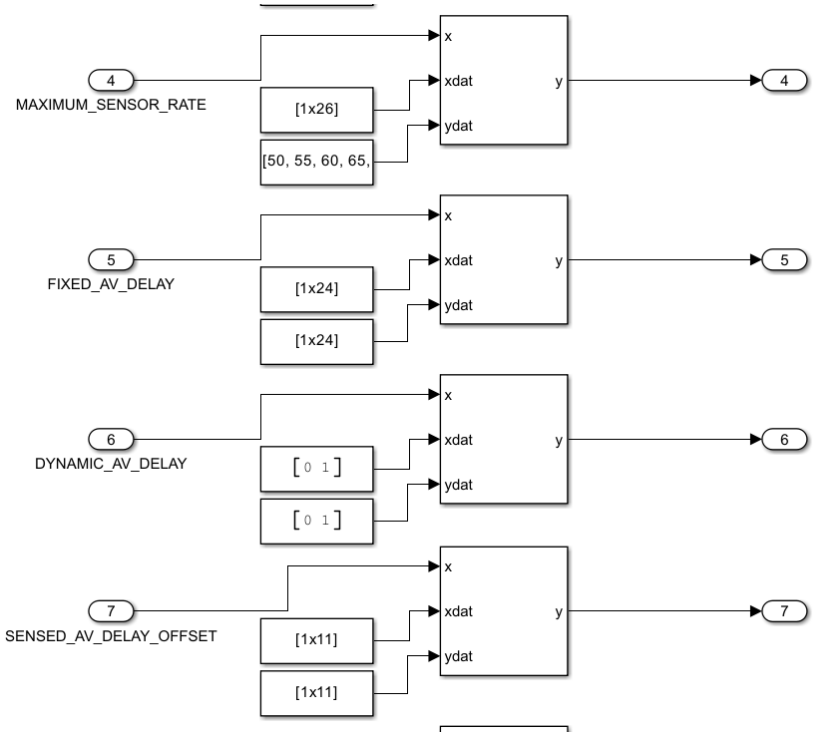
Serial communication subsystem chart stateflow for receiving data

When the signal is received, starting from the third byte, to the last one (28th), those 26 bytes contain the programmable parameters that are needed to run the pacemaker. The state on the right assigns each byte to the corresponding variable associated with it. The reasoning for having the parameters sent in 1 byte iterations (uint8) is explained below.



Send function in serial communication subsystem

As for sending a signal, that would require calling the first byte as 16 and the second as 22, and then the stateflow will flow to the send function which combines all of the byte information of the programmable parameters into a 42 byte packet. The first 26 bytes of that package contain the data of the programmable parameters, represented by 1 uint8 byte each. The last 16 bytes represent 2 double values that contains the egram data for the atrium and ventricle, used by the DCM in the real-time electrocardiogram. The reason for using 26 bytes of (uint8) is that it keeps the packet size small and keeps the system efficient. The reason this works is explained below; the function of the lookup tables.

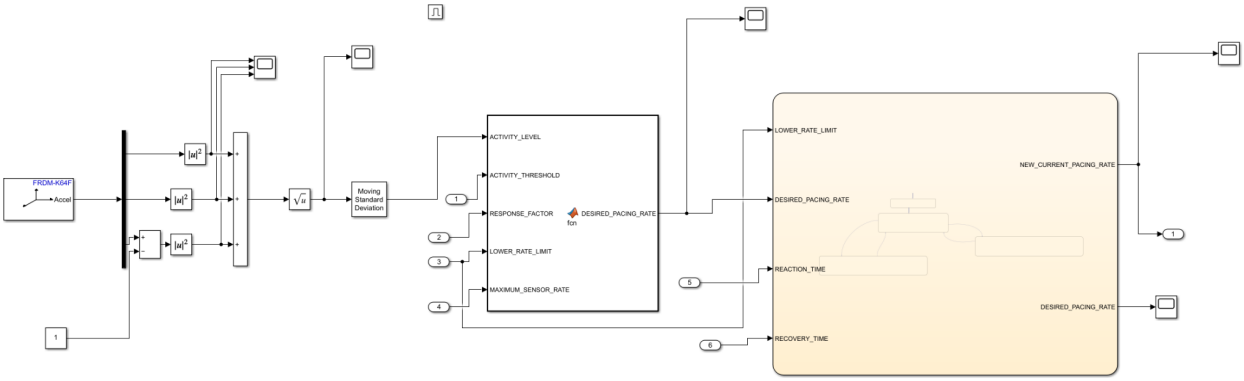


Part of the lookup tables in the serial communication subsystem that decode the byte size information into values for the programmable parameters.

As mentioned above, using single bytes to send programmable parameter information was a deliberate design choice. Essentially, it is using the fundamentals of indexing through an array for values. The DCM sends an index for each parameter and the simulink model uses lookup tables to parse through the pre-set array and find the value at the index value sent in. This minimizes the size of data we are sending in the packets and improves the efficiency of the system. Furthermore, this method doubles as a security feature as the data contained in the serial communication ports is simply comprised of index values that mean nothing without the correct array to go along with it. This level of encryption provides an extra layer of security that helps the overfall safety of our system as a whole.

2.1.6 Rate Adaptive Mode Implementation

Once of the major requirements for this stage was the functionality of rate adaptive pacing modes. They were implemented by creating a separate subsystem to follow the hardware hiding principles.



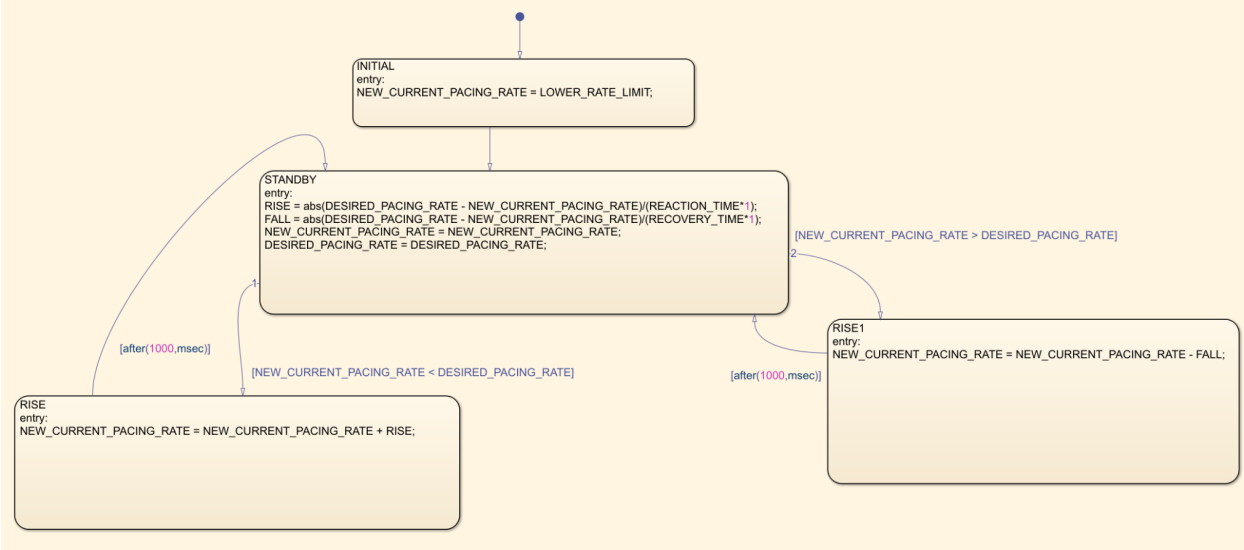
Accelerometer data processing system on Simulink

The first step to implementing the rate adaptive modes is to retrieve the accelerometer data and manipulate it in a way to analyze it, in a coherent manner. Now, the first step in processing the raw accelerometer data received form the board is to demultiplex it into x, y, and z directional components. This is done by the black bar in simulink. Then the vector components are normalized with the appropriate mathematical methods. This normalized value is then fed to a moving standard deviation block. This has the effect of smoothing out the magnitudes of the movement at each step using the standard deviation.

```
function DESIRED_PACING_RATE = fcn(ACTIVITY_LEVEL, ACTIVITY_THRESHOLD, RESPONSE_FACTOR, LOWER_RATE_LIMIT, MAXIMUM_SENSOR_RATE)
    if ACTIVITY_LEVEL >= ACTIVITY_THRESHOLD
        slope = (MAXIMUM_SENSOR_RATE-LOWER_RATE_LIMIT)/((ACTIVITY_THRESHOLD + (((1.6-ACTIVITY_THRESHOLD)/16)*(16-(RESPONSE_FACTOR-1)))) - ACTIVITY_THRESHOLD);
        b = LOWER_RATE_LIMIT - (slope*(ACTIVITY_THRESHOLD));
        if(slope*ACTIVITY_LEVEL + b > MAXIMUM_SENSOR_RATE)
            DESIRED_PACING_RATE = MAXIMUM_SENSOR_RATE;
        else
            DESIRED_PACING_RATE = slope*ACTIVITY_LEVEL + b;
        end
    else
        DESIRED_PACING_RATE = LOWER_RATE_LIMIT;
    end
end
```

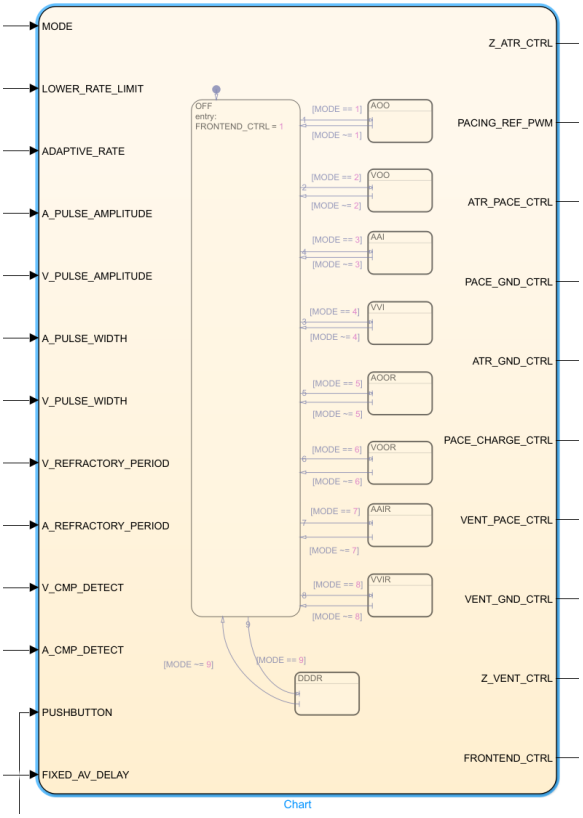
Rate response function

Then, the next main step performed by the rate response function block in the middle of the figure above. The programmable parameters set in the DCM; activity threshold, response factor, lower rate limit, and maximum sensor rate are set to be inputs for this function. The output of the function is the desired pacing rate as evaluated by the function depending on these parameters. It will essentially increase the desired pacing rate if the response factor is high when activity goes past the threshold and vice versa. The 1 constraint is that the rate should not exceed the maximum sensor rate or fall below the lower rate limit which is why they are inputs for the function as well. The response factor determines at what rate the pacing rate will increase, it determines the slope of the increase. The response factor is a value between 1-16 which corresponds to a different and increasing linear slope at each of those values at which the pacing rate will increase. The slope calculation seen in the function uses the activity level to first check if it exceeds the threshold set in the DCM, then if it has it uses the response factor to determine at what slope the pacing rate should increase. The slope and its respective y-intercept combine to make a linear system and then depending on where the activity level is (from the accelerometer) it determines what the desired pacing rate should be. With that function, the desired output rate will never go beyond the bounds set by the lower rate or the maximum which is a safety critical feature.



Simulink model for rate response

The newfound desired pacing rate is to the Simulink chart seen on the far right in the figure above. The states inside the chart can be seen in the figure right above. Along with the desire rate, the lower rate limit, reaction time and recovery time are also fed as inputs. The function of these states is to update the current pacing rate as required by the rate response function. The functionality is to increase the current pacing rate by a consistent factor at a consistent interval. The opposite also holds if the rate needs to be slowed down depending on whether the activity has stopped and the recovery interval is in effect. The reaction time is used to determine at what rate the current pace should increase with the RISE variable. The difference between the desired and current pace is taken and divided by the reaction time to obtain a factor at which to increase the rate in the RISE state seen on the left in the Simulink state model in the figure above. For the FALL state, the FALL variable is calculated in the same way as RISE but the recovery time input is used instead because of the nature of the state. The state on the right manipulates the current rate to make it drop by a consistent factor for each time step and eventually brings it to the level of the desired rate. The output of this state is the new pacing rate which is NEW_CURRENT_PACING_RATE. This output is fed to the modes stateflow chart which takes it as the adaptive rate needed to facilitate the rate adaptive modes.

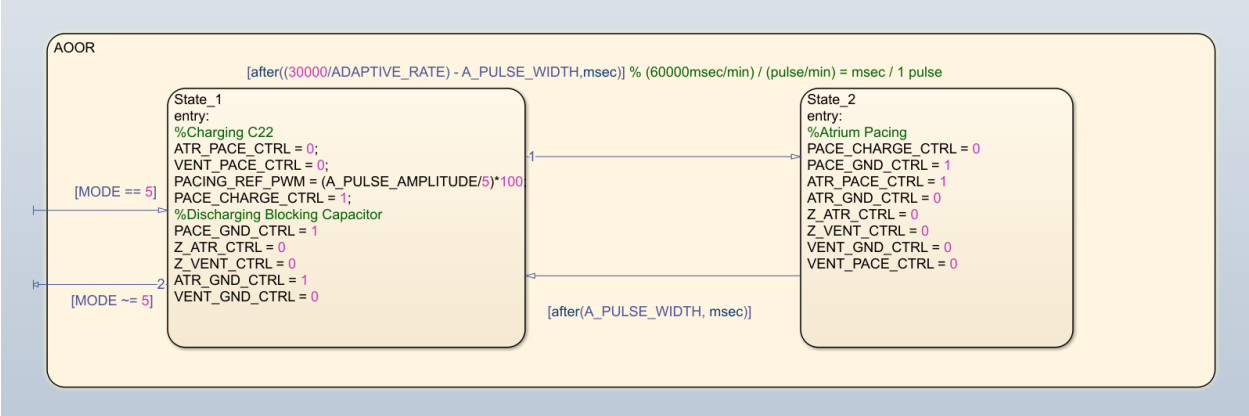


All pacemaker modes stateflow chart

The stateflow chart takes the programmable parameters from the DCM, sensing subsystem, and accelerometer data processing subsystem as inputs. Depending on the operating mode of the pacemaker, it will output 1 (HIGH), 0 (LOW), and 0-100 (PWM) for input pins on the K94F microcontroller.

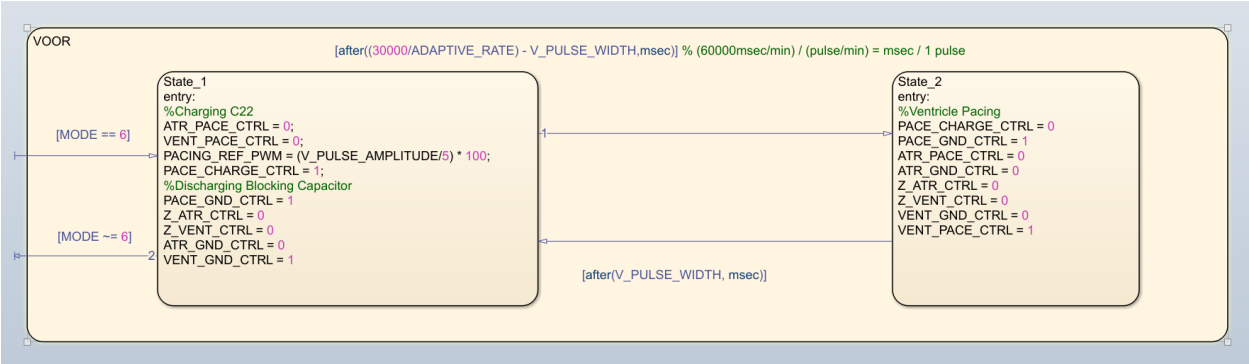
Selecting the mode is done through the DCM with each mode representing one of the 8 main pacing modes available as well as mode 9 being the DDDR mode.

As for the 4 new rate adaptive modes; their function is similar to their respective non-rate adaptive counterparts. The change lies in the pacing rate being determined by the rate response function found using the methods explained above.



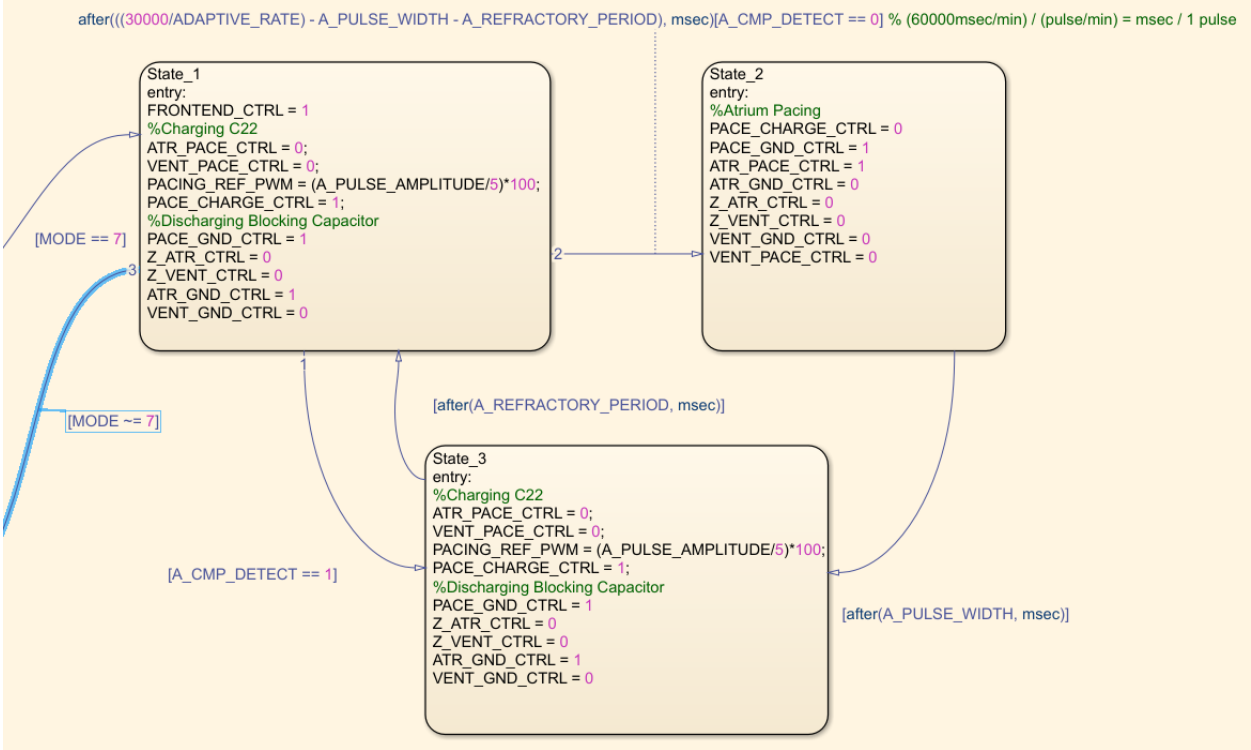
AOOR mode

AOOR has the same structure for the charging and discharging states as AOO. The difference in functionality lies in the adaptive rate variable that is sent in by the rate response function analzed above. Essentially, instead of using the lower rate limit to determine how long the time period should be between the two states, the adaptive rate is used in the calculation instead. This has the effect of shortening the time interval when the rate is higher and therefore the atrium needs to pace faster. The inverse relationship holds true as well.



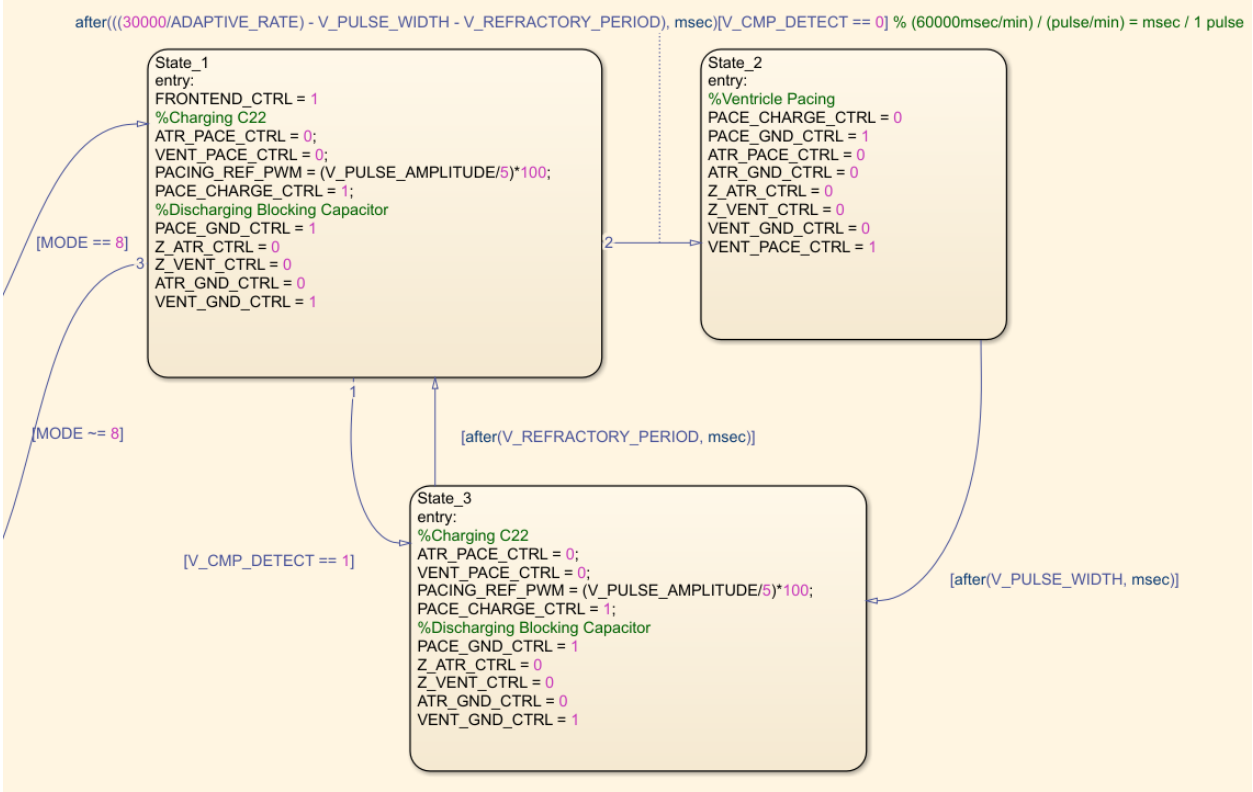
VOOR mode

VOOR works in the same manner as VOO and has the same properties explained for that state model. Once again, the adaptive rate controls the varying pacing required as per the activity detected by the rate adaptive system.



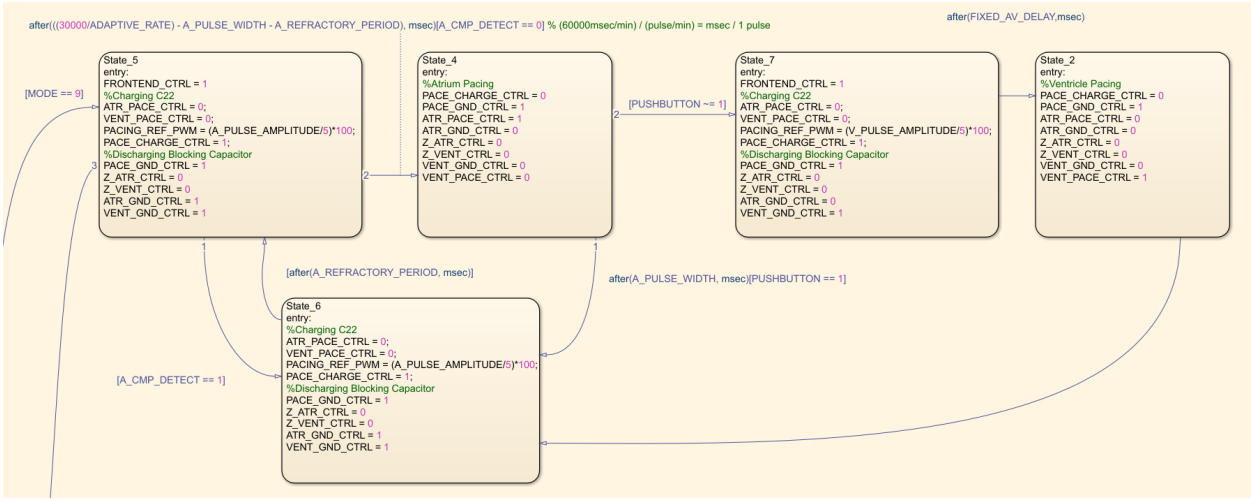
AAIR mode

AAIR has the same state flow to AAI since the functionality it is supposed to output remains the same with the addition of the rate adaptive element. Therefore, the states for charging, discharging and waiting after an atrial event are the same. The key difference once again lies in using the adaptive rate as the value for determining when to charge and discharge the capacitor opposed to using the lower rate limit for the previous AAI mode. This once again has the effect of shortening the time interval when the rate is higher and therefore the atrium needs to pace faster. The inverse relationship holds true as well.



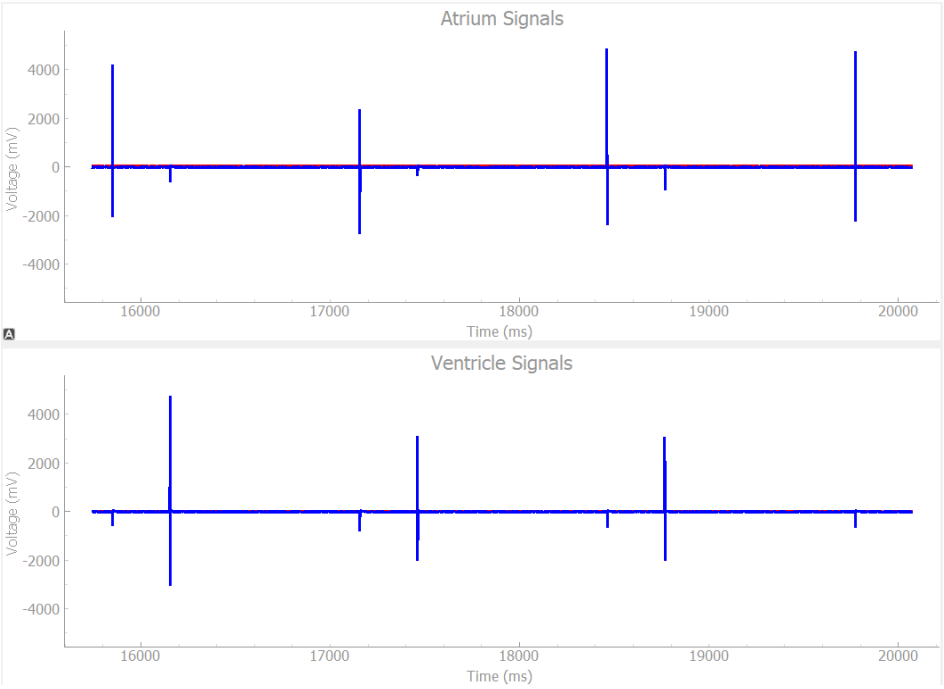
VVIR mode

VVIR, similar to the atrial modes, has the same state flow to VVI since the functionality it is supposed to output remains the same with the addition of the rate adaptive element. Therefore, the states for charging, discharging and waiting after a ventricle event are the same. The key difference once again lies in using the adaptive rate as the value for determining when to charge and discharge the capacitor opposed to using the lower rate limit for the previous VVI mode. This once again has the effect of shortening the time interval when the rate is higher and therefore the atrium needs to pace faster. The inverse relationship holds true as well.

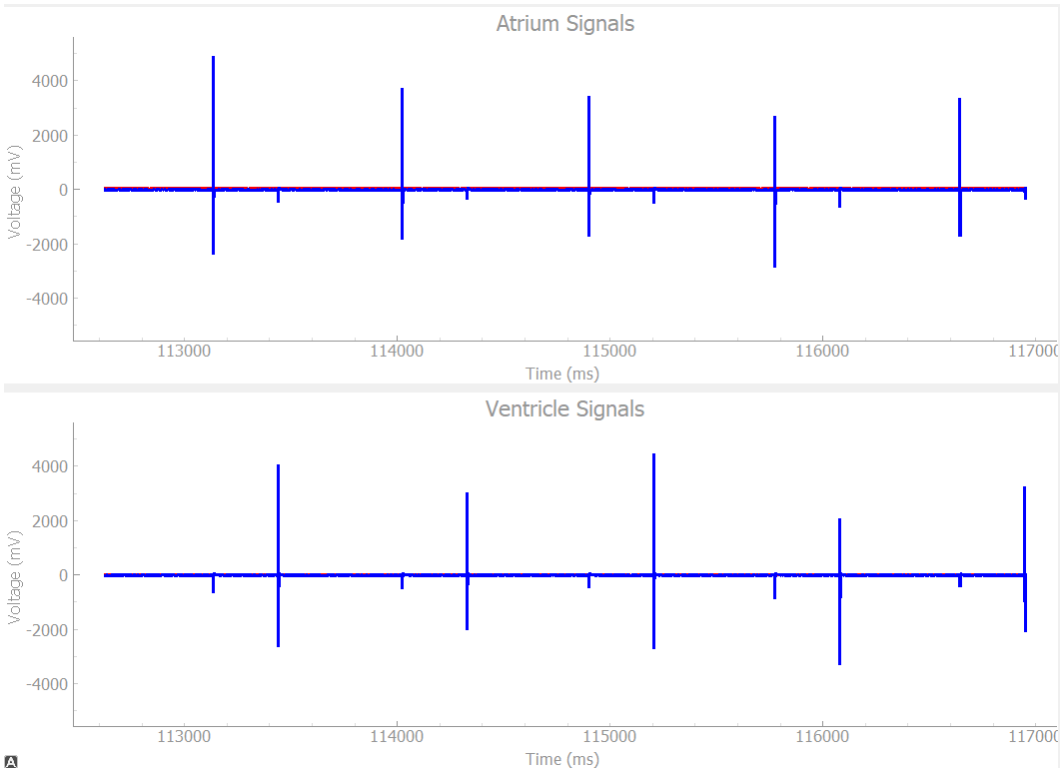


DDDR mode

This stateflow is for the implementation of the adaptive dual-pacing mode. When the pacemaker is performing in this mode, both chambers are receiving a pace however, the chambers cannot pace at the same time so there is a proper fixed AV delay programmed into the stateflow to deal with this. The delay is a programmable parameter that is variable in the DCM. Furthermore, ventricular pacing is inhibited when holding down the pushbutton as per the requirement for this pacing mode. Once the pushbutton is released then ventricular pacing will appear. The key feature of this mode is the tying together the previous XXIR modes. After the atrium pulses, the stateflow follows into the ventricle charging state and then the ventricle is pulsed only after that fixed AV delay. This is then looped back to normal functionality and the flow continues to allow for a dual-pacing mode.



Heartview output for DDDR, resting pace



Heartview output for DDDR, after activity introduced

2.2 - DCM Design

2.2.1 - General Design Choices

Theme

A dark theme with blue accents was selected as the main theme of the application. This was selected as a modern design was envisioned by the team.

Language of Choice

Python was selected as the language of choice for developing the DCM. Python offers a wide variety of libraries which can be used to help facilitate the development process. It also provides an intuitive syntax and built-in data structures to help process data and perform tasks. Python was also selected due to the team's familiarity with the language and the vast documentation available.

Repository Structure

The repository structure of this application involves main_app.py acting as the main application file, which contains the main app module. Within the same directory are folders which contain additional resources. The program_files folder consists of additional Python files to be imported as modules. These consisted of classes to be imported or global variables. The user_data folder is the directory in which the user data will be stored as a JSON file for this application. This structure was chosen to prevent the overcrowding of code into a singular Python file and to spread them out for ease of editing. The global variables are placed into separate files for easy access by importing the files rather than continuously declaring the variables for each Python file. These variables were also designed to be accessible throughout every module for ease of function.

Imported Libraries

The following table consists of all of the imported libraries used for the development of the DCM application software (these were imported using the import function in Python):

Imported modules along with their function.

Imported Module	**Function**
tkinter	Library for making GUI
customtkinter	Upgraded version of Tkinter to allow for the creation of a GUI with a modern aesthetic
json	Allows the program to save and open JSON files
os	Allows for folder navigation on the computer
time	Allows to get the current time

datetime	Gets the current date and time
numpy	Number and array processing
matplotlib	Plotting and data visualization
sys	System characteristics
serial	Serial communication
glob	Retrieve com ports
struct	Format byte data

These selected modules played a pivotal role in facilitating the development of the application. Tkinter and customtkinter allowed for an interactive UI which allowed easy creation of buttons, labels, text boxes, frames, fonts, and more.

Json is used to open, read, and save JSON files which will contain information about a user. This info contains the username, password, email, mode being paced, parameters and values, recognized devices, and history of parameters and their values.

Time was used for displaying the current date and time in the navigation bar, present on all of the screens of the application.

Datetime was used to save the history of parameter changes for modes and will be used when retrieving bradycardia and temporary reports.

2.2.2 - Global Variables

Modes and Parameters

Global variables can be accessed from any module which contains information about the modes and their parameters. These variables can be found in mode_variables.py. These variables are imported into any relevant Python files to be used. Having these variables in a separate file makes the code much more neat and organized. The following table displays all relevant variables:

Variables used for the modes and parameters and their corresponding type and function.

Variable	Type	Function
lst_parameters	list	List of all of the available parameters present on the PACEMAKER.pdf document for every mode, in order
dict_param_nom_vals	dictionary	Dictionary containing all of the different parameters listed on the PACEMAKER.pdf

		document as well as their nominal starting values. This is used to help initialize the nominal values of a newly registered user.
dict_param_and_range	dictionary	Dictionary containing all relevant parameters for AOO, VOO, AAI, and VVI and their allowed ranges. This is used to develop the parameter editing to prevent the user from selecting a value for the parameter that is not allowed
dict_modes	dictionary	Dictionary containing the four relevant modes as the keys, and the values are a list which contain indices relating to the parameters in lst_parameters. This is used for initializing the different modes for a new user by selecting the appropriate parameters for said mode
dict_modes_enumeration	dictionary	Dictionary containing the relevant modes as key and an increasing integer value, starting at 1 which represents what mode is selected. This will be used to transmit information to Simulink
dict_param_and_tolerance	dictionary	Dictionary containing the tolerance for the relevant modes for AOO, VOO, AAI, and VVI

These variables are imported to any relevant Python files for ease of access of the modes and parameters. This was kept as a global variable since it does not contain valuable information regarding how the system functions or personal user data.

Colours

Global variables representing the colours chosen for the colour palette can be found in app_colors.py. The colours are represented as hex colour codes as strings. The following table displays all the chosen colours and the variables:

Colours used in the DCM with their corresponding variable and value.

Variable	Value	Colour
bg_color	“#1A1A1A”	
gray_1	“#2A2A2A”	
gray_2	“#8f8f8f”	
gray_3	“#888888”	
blue_1	“#195FA6”	
blue_2	“#225e9c”	

blue_3	“#317ac4”	
white_1	“#D9D9D9”	
white_2	“#ffffff”	
green_1	“#3FAB4A”	
green_2	“#67AB6E”	
red_1	“#D13434”	
red_2	“#D25E5E”	
orange_1	“#D18034”	
orange_2	“#d18f52”	

These variables are imported to any relevant Python files for ease of access of the colors. This was kept as a global variable since it does not contain valuable information regarding how the system functions or personal user data.

Functions

Global functions were used in this program to allow for universal access to important functions to allow for ease of functionality the following table displays the global functions created to be imported into python files which require them:

Function	Inputs	Function	Return Type
list_serial_ports	N/A	Finds all active communication ports within the host computer	List[str] : list of all active com ports
format_data	current_mode _data (dict)	Formats the data for the current mode selected by converting the actual values to indices into a list to be sent to the pacemaker overal serial communicaton	List[int] : list of indexes representing numbers
encrypt_password	Password (str)	Takes in a password and processes it to return an encrypted verison of it for safety and security	Str : encrypted password
decrypt_password	Encrypted_pa ssword (str)	Takes in an encrypted password and returns the decrtyped verison	Str : decrypted password

List_serial_ports will return all open communication ports on the current device. This can display which port is used to connect to pacemaker and which one is for the heart. The ports are used for serial communication between the DCM and the pacemaker

Format_data will take in a dictionary of a parameter and format the data properly so that it can be sent to the pacemaker for storage

Encrypt_password and decrypt_password are used for managing user passwords through encryption to ensure that data is safe and that passwords are stored securely.

2.2.3 - Module Design

The number of modules used was limited to an attempt to keep coupling low while keeping cohesion amongst individual modules high. Most of the program lies within the main app module, and separate modules were used to help elevate the main app module.

Main App Module

The main app module can be found in the main_app.py file. This module is named DCM and inherits from its parent class, customtkinter.CTk. This is the class that allows for the creation of a screen.

Methods

Private methods of the DCM class are designed to prevent code beyond the scope of the class from accessing these methods. These methods are preceded with an underscore like the private variables. These methods are declared and used later on in the program. The following table displays the private methods, their inputs, function and output:

Private methods of the DCM class, their inputs, function, and what they return.

Class DCM(customtkinter.CTk)			
Private Method	**Inputs**	**Function**	**Return**
__init__	Self	Intialize an object of the class and its parent class, customtkinter.CTk	N/A
_create_login_screen	Self	Creates the design of the login screen and prompts user to either (1) login by entering the correct information or (2) create a new account	N/A
_create_main_interface	Self	Creates the design of the main interface and displays all buttons for modes, admin login, log out, etc.	N/A
_back_to_login	Self	Navigates from “Sign Up” screen to “Log in” screen after destroying all widgets on the current screen	N/A

_create_signup_screen	Self	Creates the design of the “Sign Up” page and prompts user to enter information to create a new account	N/A
_sign_out	Self	Logs out of current account that it is logged into and resets most of the monitored variables	N/A
_create_about_page	Self, back_to_previous_page (function) : method for creating the current page you are on	Opens up the “About” page where data about the GUI (application model number, software version, serial number, and institution name) is displayed	N/A
_create_header	Self, master (customtkinter.CtkFrame) : pass in a frame to own the widget, back_to_previous_page (function)=None : method for creating the current page you are on	Creates the display for the top of the page where the battery status, connection status, about page, time and date are found	N/A
_open_credential_prompt	Self	Creates a lavel to notify when the incorrect credentials are entered	N/A
_open_successful_register_prompt	Self	Pop-up window that is displayed when the account is successfully registered	N/A
_open_username_taken_prompt	Self	Notifies user that the username that is being used to register a new account is already taken by a previous account	N/A
_open_email_taken_prompt	Self	Notifies user that the email that is being used to register a new account is already taken by a previous account	N/A
_open_password_confirm_pad_prompt	Self	Notifies user that the password entered in the “Confirm Password” textbox and the “Password” textbox do not match	N/A

_open_admin_login	Self	Opens a window that prompts user to enter admin password to access admin privileges	N/A
_open_delete_account	Self	Opens window where user is prompt to enter admin password to delete the current account that is logged in. creates an object of the delete_account class.	N/A
_open_egram	Self, serial : serial.serialcommunication used for calling serial communication to retrieve data	Opens a window that displays the electrocardiogram by creating an object of the egram_window class	N/A
_callback	Self, *args : event arguments	Updates the UI when the permission changes between “Client” and “Admin”	N/A
_callupdate	Self, *args : event arguments	Updates function whenever the edit button is pressed or a new choice has been made from drop down menu	N/A
_monitor_connection	Self, *args : event arguments	Monitors the connection status and changes the display according to the status	N/A
_monitor_battery_level	Self, *args : event arguments	Monitors the level/percentage of battery remaining and updates display accordingly	N/A
_sign_up_check	self, username, email, password, confirm_password	Checks if the registered user is valid	N/A
_get_current_users	Self, root_dir (str) : directory to look at for saved users	Function that reads all of the user data stored in the json files	2D list with one list containing all of the usernames and the second list containing their corresponding emails

_attempt_login	self, username (str) : entered username, password (str) : entered password, all_user_data (list) : info on all available users for username and email, root_dir (str) : directory to look at for saved users	Attempts to login with a username and password	N/A
_submit_admin_password	Self	Submits the admin password from teh pop-up window	N/A
_delete_account	Self	Deletes the account of the client/user	N/A
_toggle_button	Self, btn (customtkinter.CTkButton) : button to be toggled	Toggle button that toggles between the normal and disabled state	N/A
_disable_button	Self, btn (customtkinter.CTkButton) : button to be disblaed	Disables all the buttons that are only available to the admin when in client view	N/A
_enable_button	Self, btn (customtkinter.CTkButton) : button to be enabled	Enables the buttons that were originally disabled in client view when logged into admin view	N/A
_get_parameter_data	Self, values (list) : updates _updated_parameter_values with a new list of values for each parameter from the scroll_parameters_frame class	Function to retrieve data from the scrollable frame class with the sliders to bring it into the main app class	N/A
_init_parameters_on_mode_selection	Self, values (list) : values of parameters for a mode, values_indexed (list) : values of parameters for a mode but indexed relative to its position in a list	Sets the saved parameter values when mode is selected	N/A
_stop_button_cmd	Self	Command when the “Stop” button is clicked. Stops pacing	N/A

_start_button_cmd	Self, selected_mode (str) : selected pacing mode	Command when the “Start” button is clicked. Starts pacing.	N/A
_pacing	Self, selected_mode (str) : selected pacing mode	Command to send the current user saved parameters to pacemaker	N/A
_pacing_on_connection	Self, data (list[int])	Sends a packet to pacemaker upon DCM detection of pacemaker connection	N/A
_stop_pacing	Self	Stop pacing pacemaker	N/A
_verify_data_on_pacemaker	Self	Checks that the data on pacemaker matches the current data selected	N/A

The methods _create_login_screen, _create_main_interface, _back_to_login, _create_signup_screen, _sign_out, and _create_about_page are methods used for page navigation within the application. These methods are called and will destroy any outstanding widgets and create new widgets to represent a new page.

The _create_header method will create the header seen on all pages of the DCM application. The header includes the connectivity status of the DCM to the device, the battery status, the current date and time, and a button for navigating to the About page.

The methods _open_credential_prompt, _open_successful_register_prompt, _open_username_taken_prompt, _open_email_taken_prompt, _open_password_confirm_bad_prompt, _open_admin_login, _open_delete_account, and _open_egram are used for prompting the user. Some are used to prompt the user when a bad credential is entered during the registration or sign-in process. Some are used to create pop-up windows to allow for the user to interact with.

The methods _callback, _callupdate, _monitor_connection, and _monitor_battery_level are used to monitor tkinter variables. These methods are run whenever their corresponding variable gets updated by writing to it. All of these methods require *args to be an argument as there are many implicitly passed-in arguments during the invoking of these methods. Once the monitored variable is modified, these methods will run and update the DCM in a corresponding manner, depending on the variable

_attempt_login is used to check if the credentials entered from the sign up screen and comparing them with the data of registered users to see if they are in the database. The _sign_up_check method is used to see if any entered credentials are already taken by a user in the system to avoid users of the same username and email from registering for an account.

_get_current_users will retrieve all existing users and their associated email address which is then used in the _attempt_login and _sign_up_check methods to see if the credentials the user has entered is in the system.

_submit_admin_password, _delete_account, and _get_parameter_data are methods which allow for communication between other modules. These methods are passed in as arguments during the creation of objects for these modules and are then called when appropriate to communicate information back to the main DCM application.

_pacing, _pacing_on_connection, _stop_pacing, _verify_data_on_pacemaker all use serial communication to communicate to the pacemaker through the DCM. they are used to send and receive packets to ensure that the data sent and/or received is correct to account for any safety/security concerns

Variables

Private variables were declared to prevent access to these variables from outside the class's scope. The names of these variables are preceded with an underscore. These object variables are used for reference later in the program. The private object variables and their method of declaration are as follows:

Private object variables along with their purpose/function.

Private Variable	**Purpose/Function**
Constructor Method (__init__)	
_perms	Monitored string variable that represents the current permission, either as “Client” or “Admin”
_can_edit	Monitored boolean variable that represents whether the user can edit the parameters
_mode_choice	Monitored string variable that represents the current selected mode from the drop down menu
_updated_parameter_values	Initially None, is a list containing the current parameter values for the selected mode
_updated_parameter_values_indexed	Initially None, is a list containing the current parameter values for the selected mode but its index relative in an array
_saved_parameter_values	Initially None, is a list containing the saved parameter values for the selected mode
_saved_parameter_values_indexed	Initially None, is a list containing the saved parameter values for the selected mode but its index relative in an array
_serPacemaker	Initially None, is an object of SerialCommunication and is used for

	communication to the pacemaker through the DCM
_connected_status	Status of the connection of pacemaker to DCM
_battery_level	Current battery status of the device
_all_battery_statuses	List of all battery statuses: “BOL”, “ERN”, “ERT, “ERP”
_current_user	Current user that is logged in
_admin_password	String representing the admin password
_root_dir	String representing the root directory for where the user data is stored
_toplevel_window	Represents whether a toplevel window is open
_egram_window	Represents whether the egram window is open
_about_info	Data outlined on the “About” page
Login Screen Method (_create_login_screen)	
_frm_login_screen	Window that displays the login info and the inputs the user must enter to log into their account
_txtbx_username	Textbox for user to enter username
_txtbx_password	Textbox for user to enter password
_forgot_button	Button for user to click if they forgot their password
_signup_button	Button to create a new account
Main Interface Method (_create_main_interface)	
_frm_main_interface	Window that displays the main interface of the DCM
_btn_admin	Button that allows user to log into admin and gain admin credentials
_btn_run	Button to run the mode that is set
_btn_stop	Button to stop the mode that is currently running
_btn_delete	Button to delete account
_btn_bradycardia_report	Button to display the report for bradycardia

_btn_temporary_report	Button to display the temporary report
_perm_label	Label that displays the current permissions of the UI, client or admin
_current_mode_lbl	Label to display the current mode being paced
_frm_scroll_parameters	Scrolling frame to display all parameters and their values for a mode for review and modiciation
_btn_edit	Button to edit the settings and modes
_combobox_select_mode	Drop-down menu to select the desired mode
_btn_show_egram	Button to display electrocardiogram
_btn_verify	Button to check if parameters on pacemaker match the current selected parameters
_lbl_verify	Label to be updated based on the status of the parameter verification
Sign Up Screen Method (_create_signup_screen)	
_frm_signup_screen	Pop-up screen for the “Sign Up” page
_txtbx_email	Textbox for user to enter email address
_txtbx_username	Textbox for user to enter username
_txtbx_password	Textbox for user to enter password
_txtbx_confirm_password	Textbox for user to enter password again (to confirm)
About Screen Method (_create_about_page)	
_frm_about_screen	Pop-up screen for the “About” page
Header Creation Method (_create_header)	
_lbl_connected_status	Label for displaying the current connection status of the DCM to the device
_lbl_battery_status	Label for displaying the current battery status of the connected device
_btn_about_page	Button to display the “About” page

Many of the variables that are of the class customtkinter are defined as variables as they are reference later on in the program. This allows for these widgets to be configured when appropriate to update the UI.

Monitored variables are used to update the state of the DCM whenever these variables are modified. This allows for the DCM to respond quickly and efficiently. Monitored variables were selected as the best method to invoke a quick change in the DCM as its efficient and does not require a manually programmed loop to watch for changes in the variables.

There are no public variables within this class as use of these variables beyond the scope of this class were unwanted and unnecessary.

User Module

The user module can be found in the user_class.py file. This module is named user and provides the necessary methods to create a user. This class contains public accessor and mutator methods to update the user private variables and retrieve them for use. This class also contains methods for saving the data and loading the data from a json file. A json file was selected as the format of choice for saving a user because json files allow to easily save class objects as dictionaries and load them.

Methods

This class contains no private methods and only consists of public methods to allow for access of the class contents from outside the scope of the class. These methods are used in other modules in the application to access user data or mutate variables. The following table displays the methods, their inputs, function and output:

Public methods and their corresponding inputs, function, and what they return.

Class user			
Method	**Inputs**	**Function**	**Return**
__init__	Self, username (str) : the username, password (str) : password of the user, email (str) : email of the user, current_mode (str) : current mode being paced for this user, existing_mode_data (dictionary) : dictionary containing the modes, their parameters and the values, recognized_devices (list) : contains information of recognized devices for the user, mode_parameter_history (list) : contains a history of all saved values for all	Intialize an object of the user class. If this is a newly registered user, then it will create a new data sheet using the four modes, their parameters and their nominal values. This will also be called if a returning user logins and will load all their data	N/A

	modes, parameters and values and when they were saved		
save_to_json	Self , str_root_dir (str) : root directory of the saved user data	Saves an object of the user class to a json file as a dicitonary	N/A
delete_account	Self, str_root_dir (str) : root directory of the saved user data	Deletes the json file of the current user	N/A
load_from_json (classmethod)	Cls (class) : implicitly passed in class of user, dict_user (dicitonary) : loaded dictionary from json file	Load the user from their saved json file as an object of class user	Object of class user
get_username	self	Gets the username	_username (str) : username of user
get_password	self	Gets the password	_password (str) : password of user
get_email	self	Gets the email	_email (str) : email of user
get_current_mode	self	Gets the current mode paced	_current_mode (str) : current mode paced
get_all_mode_data	self	Gets all current mode and parameter data	_all_mode_data (dictionary) : all mode and parameter data
get_all_recognized_devices	self	Gets all recognized devices	_recognized_devices (list) : list of recognized devices for the user
get_mode_parameter_history	self	Gets the history of all modes and parameters	_mode_parameter_history (list) : contains dictionary of mode data and when it was changed
set_all_mode_data	Self,	Sets all mode and	N/A

	updated_all_mode_data	parameter data and logs it in ths history	
set_username	Self, new_username	Sets a new username	N/A
set_password	Self, new_password	Sets a new password	N/A
set_current_mode	Self, new_mode	Sets a new current mode paced	N/A
add_new_device	Self, device	Adds a new recognized device	N/A
get_formatted_data	Self	Gets the current mode parameter data and formats it to be sent to the pacemaker through serial communicaiton	Formatted_data (list[int]) : formatted data list that can be interpreted by the pacemaker

The methods save_to_json, delete_account, and load_from_json are used to manipulate and modify the json files for a specific user. A json file was selected as the best way to save a class object as there are built in methods to load json files directly as a dictionary so retrieving information is facilitated. The save_to_json method will save all of the private variables associated with a user to a json file in the form of a dictionary so that each private variable can be easily accessed. The delete_account method will search the root directory inputted and delete the json file of the current user object and will delete the class instance of itself. The load_from_json method is a class method, meaning it can be invoked upon declaration of the class object. This method is used to load all the data from the json file of a saved used into a class object to be used later on in the application and will invoke the constructor method automatically.

The accessor methods are used to access the private variables of the class externally to avoid calling the variables directly. This method was selected as it is alot more secure than reading the data straight from the variable as that might cause accidental modification of the variable.

Mutator methods are used to modify the private variables of this class. The public mutator methods allow for ease of access for modification of these variables and provide a convenient and consistent method of modifying these variables. These were selected as to avoid calling the variable itself to avoid any potential dangers associated with it.

The get_formatted_data method will return a list of formatted parameters that the pacemaker can interpret which can then be convered into a selected mode and its associated parameter for pacing the heart.

Variables

Private variables were declared to prevent access of these variables from outside of the scope of the class. The names of these variables are preceded with an underscore. These are object variables which are used for reference later on in the program. This module contains only private variables for use and no public variables. The private object variables as well as their method of declaration are as follows:

Private object variables along with their corresponding purpose and function.

Private Variable	**Purpose/Function**
Constructor Method (__init__)	
_username	Holds the username
_password	Holds the password
_email	Holds the email
_recognized_devices	Holds the list of all recognized devices
_current_mode	Holds the current mode being paced
_all_mode_data	Holds a dictionary of data for the current save settings for modes and parameters
_mode_parameter_history	Holds the list of history of past modes and parameter values

These variables are used to represent any necessary information associated with a user. The _username, _password, and _email variables are used for login and registration of the user to ensure that the entered credentials match. The _recognized_devices, _current_mode, _all_mode_data, and _mode_parameter_history are used for DCM function for pacing and modification of the modes.

Successful Registration Prompt Module

This module can be found in the app_widgets.py file. This class is used to create an object that inherits from the customtkinter.CTkToplevel class to create a pop up window in addition to the current main application screen. The purpose of this class is to notify the user when they have successfully registered for an account from the sign up screen.

Methods

This class only contains the constructor method and no other additional methods.

Constructor method and the inputs, function, and what it returns.

Class successful_register_prompt(customtkinter.CTkToplevel)			
Method	**Inputs**	**Function**	**Return**

__init__	self	constructor method to initialize the the popup screen to notify that a new user has been successfully registered	N/A

The constructor method intializes the window and any associated widgets present on this window. This top level window is focused until closed.

Variables

There are no private or public variables for an object of this class as it is unnecessary as this class is only for notifying the user of when their account is successfully registered. This module does not communicate between the other modules of the application.

Admin Login Module

The admin_login module can be found in the app_widgets.py file. This class inherits from the class customtkinter.CTkToplevel. The purposeof this module is to create a separate pop up window to allow for the user to log in as an admin to access the unrestricted version of the DCM main interface. When active, the admin login screen will be focused until the user enters the correct password or until exited.

Methods

This module contains only private methods and no public methods. Public methods were omitted as there is no need for the module to communicate beyond the scope of the class itself.

Methods along with their corresponding inputs, function, and what they return.

Class successful_register_prompt(customtkinter.CTkToplevel)			
Method	**Inputs**	**Function**	**Return**
__init__	Self, submit_admin_password (function) : to send information from this class to the main application class, admin_password (str) : the admin password of the program	Initialize the pop up window and any associated widgets with this screen	N/A
_send_password	Self, entered_password (str) : password from the textbox	Compares the entered password and the admin password. If they match, a prompt is send to the main application module and the	N/A

		permission is changed, if not it will prompt the user	
_incorrect_password_prompt	self	Creates a prompt for the user when the incorrect admin password is entered and attempted	N/A

The _incorred_password_prompt method is called whenever a bad admin password is attempted. This will notify the user that the password is incorrect.

_send_password is used to communicate with the main DCM application module to allow for the module to know when the user requests to be logged in as an admin. This will invoke an update in the permissions of the main application module and the UI will update to the new permissions

Variables

This module contains private variables and no public variables. External use of these variables is not necessary as these variables are only used within the scope of the module.

Private variables along with their purpose/function in the constructor method.

Private Variable	**Purpose/Function**
Constructor Method (__init__)	
_label	Label for the title of the page
_get_admin_password	Function to send an invoke to the main application module
_admin_password	Stores the admin password of the application

_get_admin_password stores a method that was passed in during the object creation. This method is from the main DCM application module and allows for this module to communicate with the other module when the entered admin password is correct.

_admin_password stores a string of the correct admin password and will be checked whenever the user attempts the entered admin password.

Delete Account Module

The delete_account module can be found in the app_widgets.py file. This class inherits from the class customtkinter.CTkToplevel. The purpose of this module is to create a separate pop up window to allow for the user to log in as an admin to access the unrestricted version of the DCM main interface. When active, the delete account pop up screen will be focused until the user enters the correct admin password or until exited.

Methods

No public methods are used in the creation of the delete_account module. Private methods are used as this module does not need to called from beyond the scope of this class to invoke function calls.

Methods and their corresponding inputs, function, and what they return in the class called admin_login.

Class admin_login(customtkinter.CTkToplevel)			
Method	**Inputs**	**Function**	**Return**
__init__	Self, submit_delete_account_confirm (function) : to send information from this class to the main application class, admin_password (str) : the admin password of the program	Initialize the pop-up window and any associated widgets with this screen	N/A
_send_confirmation	Self, entered_password (str) : the entered admin password in the textbox	Compares the entered password and the admin password. If they match, a prompt is sent to the main application module and the current logged-in user will be deleted, if not it will prompt the user	N/A
_incorrect_password_prompt	self	Creates a prompt for the user when the incorrect admin password is entered and attempted	N/A

The _incorrect_password_prompt method is a command called whenever an incorrect admin password is entered and attempted. This will prompt and notify the user that the incorrect admin password has been entered.

The _send_confirmation method will communicate with the main DCM application module to notify the DCM that the user has requested to delete their account. The DCM will then execute any necessary steps to end the session for this user and delete their records from the system.

Variables

This module contains private variables and no public variables. External use of these variables is not necessary as these variables are only used within the scope of the module.

Table 13. Private variables and their function/purpose in the constructor method.

Private Variable	Purpose/Function
Constructor Method (__init__)	
_label	Label for the title of the page
_submit_delete_account_confirm	Function to send an invoke to the main application module
_admin_password	Stores the admin password of the application

_submit_delete_account_confirm holds a method from the main DCM application module. This method will be called whenever the correct admin password is entered and compared with _admin_password. This allows for this module to communicate effectively with the main DCM application module.

Scrolling Programmable Parameters Frame Module

The scroll_parameters_frame module can be found in the app_widgets.py file. This class inherits from the class customtkinter.CTkScrollableFrame. The purpose of this module is to create a scrollable parameters frame widget in the main interface. Additional widgets are placed onto this frame to represent the programmable parameters and the ability to modify them for a selected mode. This was chosen to be a scrollable frame as this allows for easy additions of new pacing modes which require more programmable parameters than what is needed at the moment.

Methods

Two methods are private methods. One public method has been added to allow for the main DCM module to call this function to initialize data upon mode selection.

Methods and their corresponding inputs, function, and what they return in the scroll_parameters_frame class.

Class scroll_parameters_frame(customtkinter.CTkScrollableFrame)			
Method	**Inputs**	**Function**	**Return**
__init__	Self, master, current-mode_data = None, current_mode = None, send_data_func = None, **kwargs	Initializes the scrollable frame widget and any corresponding labels and sliders depending on the parameters required for the selected mode. This is where the user will be able to review and modify the programmable parameters	N/A
_update_changes	self	Sends updated values for a mode representing each	N/A

		parameter to the main application module	
_init_parameters	self	Sends updated values for a mode representing each parameter to the main application module	N/A
Init_parameters_on_mode_selection	Self, current_mode_data (dict) : dictionary of parameters for a mode	Formats the data to be sent back to the main app module that is formatted to be ready to be sent to the pacemaker	N/A

The _update_changes method is used to communicate with the main DCM application module by sending updated values for the parameters of a mode whenever the slider is moved. The slider was chosen as the method of updating the parameters as it was the easiest to control the values that a user was able to select for a given parameter. This avoided the need to alert the user if they ever inputted a value that was outside of the specified ranges and increments. The slider ensured that all values will be within the appropriate ranges and fall under the correct increments.

The _init_parameters and init_parameters_on_mode_selection methods are used for sending and formatting parameter data back to the main app module, which can be ready to be sent to the pacemaker when needed.

Variables

No public variables are included in this module as reading and writing to these variables externally is not required. This will help keep cohesion high within this module as it eliminates the direct need for external code to directly modify these variables. This module only contains private variables.

Private variables and their purpose/function in the constructor method.

Private Variable	**Purpose/Function**
Constructor Method (__init__)	
_current_mode_data	Stores the current mode selected from the drop down menu from the main interface screen of the main application window
_send_data_func	Stores a function to send data to the main application window of the mode and the parameter
_init_data_func	Stores a function to send data to the main application window of the mode and the parameter on initialization of the method

_parameter_value_list	List containing values of the parameters for the selected mode
_parameter_value_list_indexes	List containing the index of values for parameters for the selected mode, formatted to be sent to the pacemaker
_parameter_sliders	List containing objects of the class customtkinter.CTkSlider to allow for the user to use a slider to modify values of the programmable parameters for a given mode
_parameter_values_label	List containing objects of the class customtkinter.CTkLabel to display the current slider value of the programmable parameter for a given mode

The value of these variables changes depending on the selected mode the user wishes to review and/or modify. These variables help display the parameter information in an efficient way without harding coding each parameter in. For the selected mode of the user, this will show the parameter name, the slider to allow for modification within the specified range, and the current slider value.

Electrogram Module

The egram_window module can be found in the app_widgets.py file. This class inherits from the class customtkinter.CTkToplevel. The purpose of this module is to create a pop up window that will allow for live electrogram data readings to be displayed to the user. Currently the module is not yet functional as there is no electrogram data to display at the current moment.

Methods

This module only contains the constructor method and no other public or private methods. This module does not yet exhibit the need to process any data electrogram data.

Method(s) and their corresponding inputs, function, and what they return in the egram_window class.

Class egram_window(customtkinter.CTkToplevel)			
Method	**Inputs**	**Function**	**Return**
__init__	Self	Intializes an object of the class which inherits from the class customtkinter.CTkToplevel to create a pop up window	N/A

Variables

This module contains a singular private variable and no public variables as this does not need to explicitly communicate beyond the scope of the module.

Private variable(s) and their putpose/function for the constructor method.

Private Variable	Purpose/Function
Constructor Method (__init__)	
_ser	Stores an object of SerialCommunication which is used to read data from the pacemaker
_x	X axis of the plot in time
_atriumECG	Y axis values for the atrium ecg signals from the pacemaker
_ventricleECG	Y axis values for the ventricle ecg signals from the pacemaker
_fig	Plot figure
_ax1	Subplot 1
_ax2	Subplot 2
_canvas	Canvas to be embedded into tkinter frame
_ani	Animation to run and update the plot

These variables in the Electrogram display module are all used in plotting the electrogram data received from the pacemaker in real time. It displays a figure with two subplots, representing the real time plots of the atrium and the ventricle.

Serial Communication Module

The Serial Communication (SerialCommunication) module can be found in the serialcomm.py file. The purpose of this module is to facilitate all serial communication between the DCM and the Simulink file, including port detection and utilization, and packet formation, sending, and receiving. The serial communication module utilizes pyserial and struct libraries for packet communication, and packet formation and deformation, respectively. This design decision was made to group related functions from each library into more concise functions within one file (serialcomm.py).

Methods

The Serial Communication (SerialCommunication) module contains no private methods and only consists of public methods to allow for access of the class contents from outside the scope of the class. The following table displays the methods, their inputs, function and output:

Class SerialCommunication			
Method	**Inputs**	**Function**	**Return**
__init__	Self, port, baudrate, timeout	Initializes an object for serial communication between the port inputs, and using the baudrate and timeout inputs	N/A
open_serial_connection	self	Opens the serial connection, utilizing pyserial specific initialization variables	N/A
close_serial_connection	self	Closes the serial connection	N/A
send_packet	self, values	Opens the serial connection, sends the values input packet, and closes the serial connection	N/A
receive_packet	self	Opens the serial connection, reads and translates the information inside the packet pertaining to patient parameters, and closes the serial connection	Tuple : Returns indices correspondi ng to select values for parameters
get_egram_data	self	Opens the serial connection, reads and translates the information inside the packet pertaining to electrocardiogram data, and closes the serial connection	Tuple : returns the current electrocardi ogram reading for ventricle and atrium

Variables

The Serial Communication (SerialCommunication) module only contains private variables, as reading and writing to these variables externally is not required, promoting high cohesion between files as it eliminates external code that directly modify these variables. Highlighted below is a table summarizing the variables created within this module:

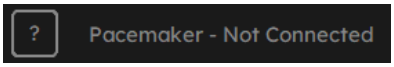
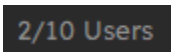
Private Variable	Purpose/Function
Constructor Method (__init__)	
port	Communication port name to transmit data for

	serial communication
baudrate	Default baudrate for serial communication
timeout	Timeout (seconds) for serial communication
ser	Variable that inherits pyserial properties needed for serial communication. Refer to pyserial documentation for details on pyserial specific initialization variables.
packet_size	Size of the packet being formatted, received and sent
packet_format	Array of data types, in order from head to tail, of the packet information needed to read

2.2.4 - Final Application

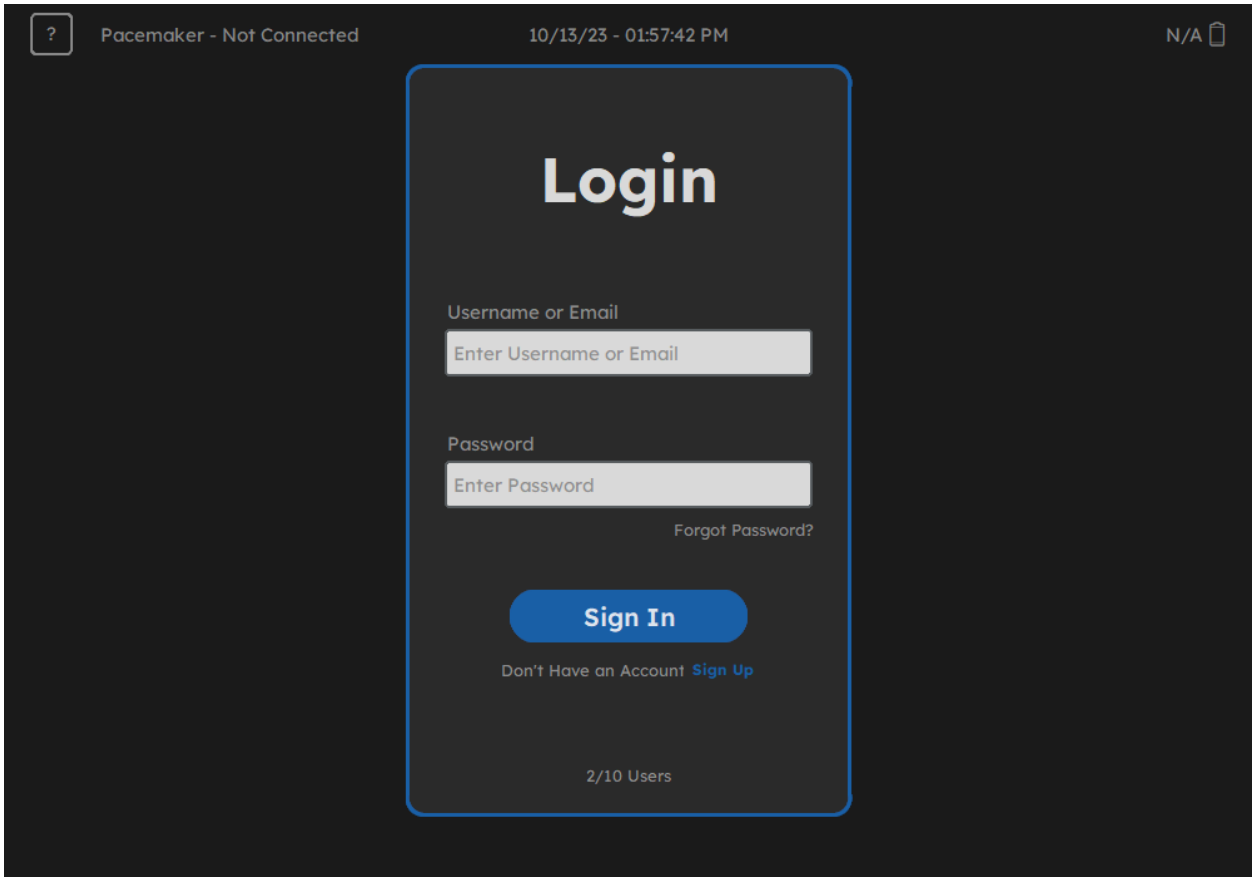
2.2.4.1 - General Design Choices

General Design Features.

Features Added	Explanation	Visual Aid
Current date and clock visual prompt.	This feature creates a more user-friendly environment.	 *Date and Clock Visual Prompt.*
Visual prompt that indicates whether the DCM and device are communicating (connected).	Essential aspects of the user interface (3.2.2.4 in PACEMAKER) make this feature a requirement. It is important for a user to see this information, regardless of being in a registered account, so all external users do not need backend inspection if this information is needed.	 *Device and DCM communication visual prompt.*
Battery status visual prompt.	This feature is important for future implementations of magnet testing (3.7 in PACEMAKER) in order to let users know the battery state of the device without backend inspection.	 *Battery status visual prompt.*
Active users are tracked and visually indicated as a fraction	This feature allows the user and/or physician to be able to	

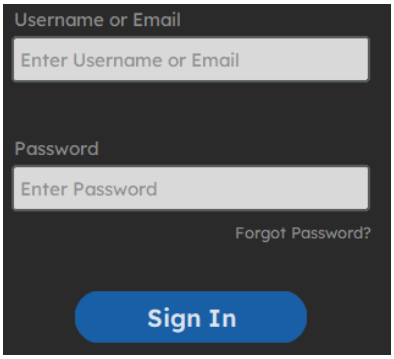
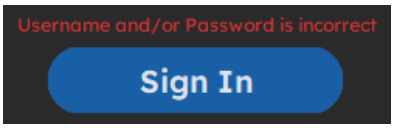
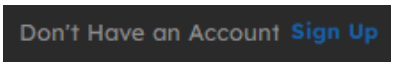
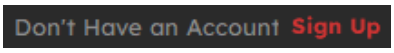
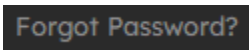
(X/10) in the login screen. (10 maximum users)	keep record of the number of users, in order to promote a more user-friendly environment that does not require backend inspection for this information.	*Active users visual prompt.*

2.2.4.2 - Login Screen Design Choices

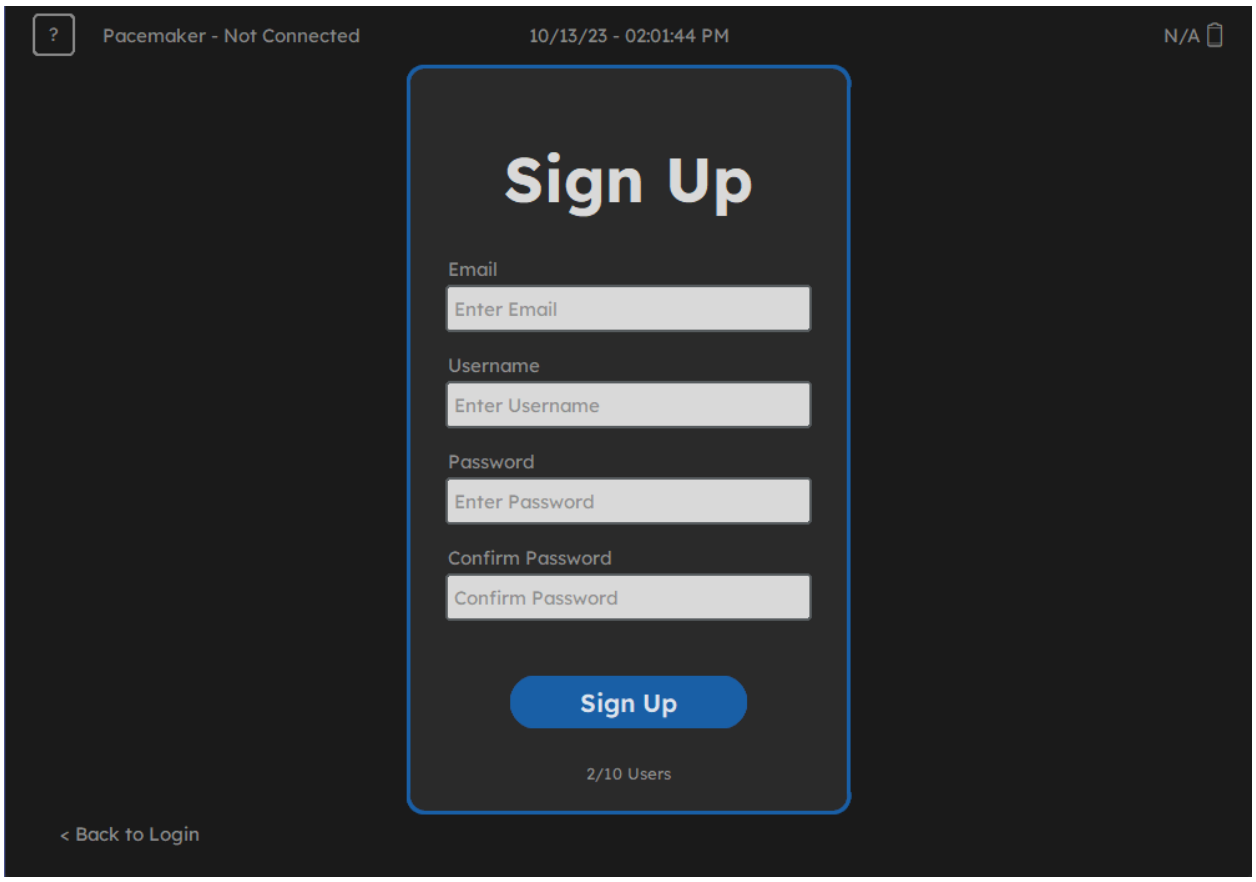


Login Screen

Login Screen Design Features

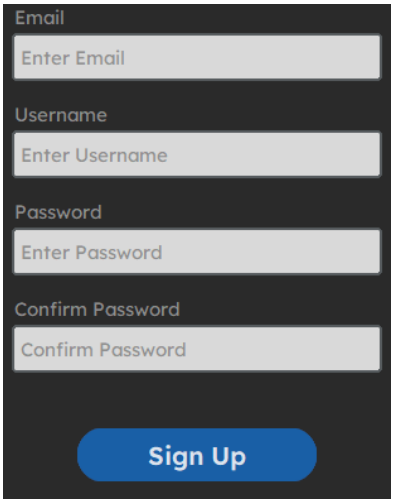
Features Added	Explanation	Visual Aid
The option to input a username and password into correlating textboxes, and can enter their credentials by pressing the sign in button or pressing enter to log in.	Assignment 1 gives the requirement to log in as an existing user. For confidentiality/privacy reasons, it is important for specific users to be able to log in to view their own pacemaker data only.	 *Username and password sign in inputs.*
The user is notified if incorrect credentials are entered and attempted.	This feature promotes a more user-friendly environment, specifically differentiating the possibility of other back end issues that may cause the user to not be able to log in versus incorrect credentials.	 *Username and/or password is incorrect notification.*
A sign up button, giving a new user the option to sign up for a new account, using the sign-up button.	Assignment 1 gives the requirement to give a user the ability to register a new account. The sign up button provides a user friendly way to navigate to registration within the main application.	 *Sign up for an account (Log in screen) button*
User cannot sign up for an account when there is a maximum amount of users, indicated by a red sign up button that is not clickable.	Assignment 1 gives the requirement to store a maximum of 10 users locally. This feature allows the user to adhere to these requirements, stopping any front-end bugs from overflow.	 *Maximum users met sign up button update*
Forgot password button, allowing a user the option to reset their password.	A user may forget their password and need to access essential information. This is a user friendly method of supporting account changes.	 *Forgot password button*

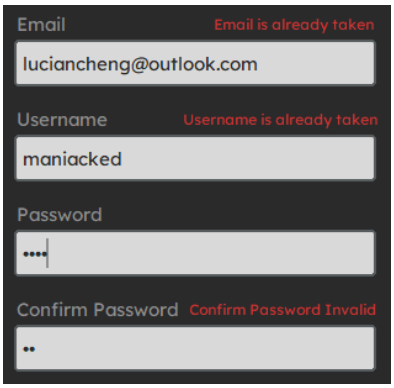
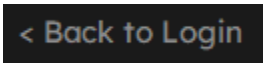
2.2.4.3 - Registration Screen Design Choices



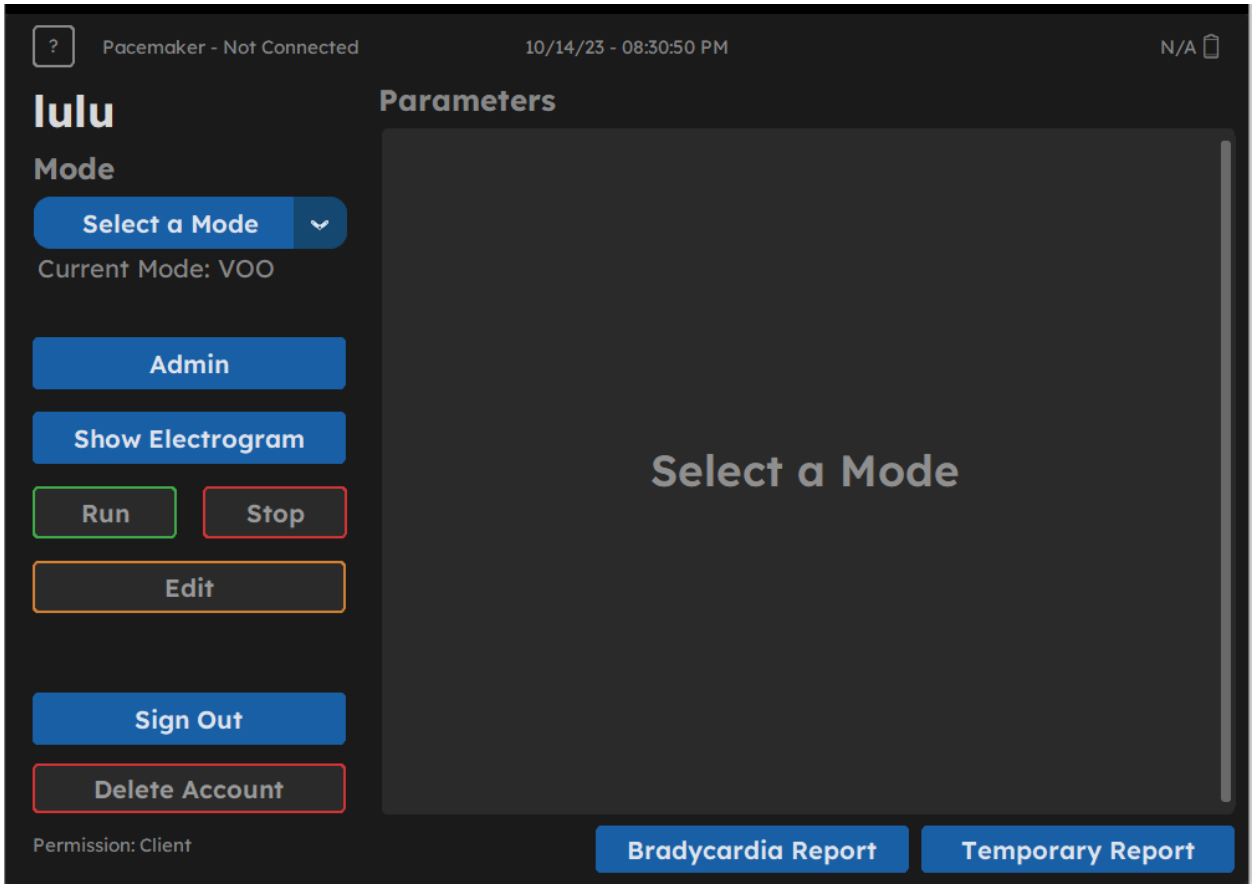
Register Screen

Registration Screen Design Features

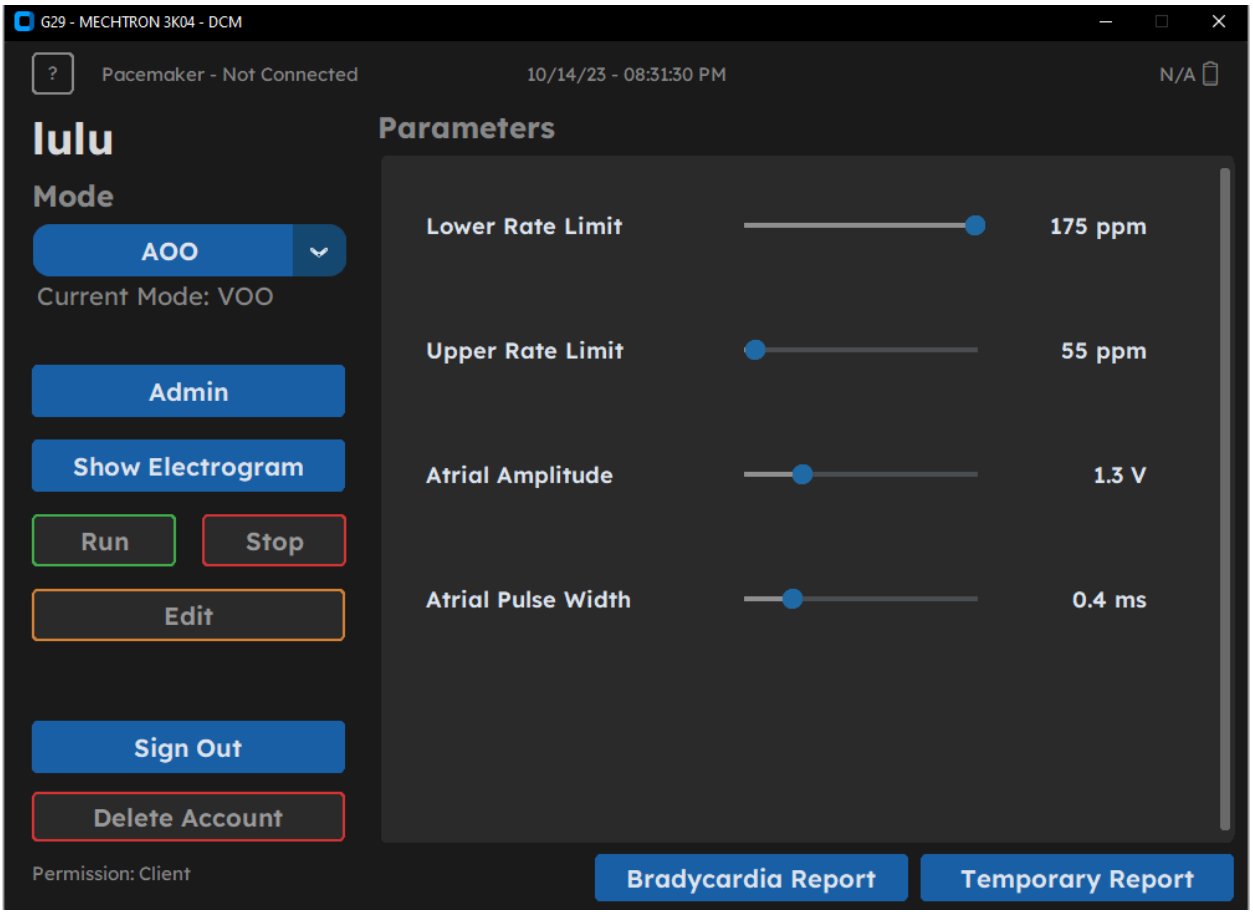
Features Added	Explanation	Visual Aid
The option to input a new username, email, password and confirm password into correlating textboxes, and can enter their credentials by pressing the sign-in button or pressing enter to log in.	Assignment 1 gives the requirement to give a user the ability to register a new account. The addition of confirmed password is important for users to double check they inputted their intended password.	 *Username, email, password, and confirm password sign up*

		inputs.
The user is notified if credentials, such as email and username, are already taken by an existing account. As well, the user is notified if the password and confirm password entry do not match.	This feature promotes a more user-friendly environment, allowing the user to troubleshoot registration issues and stop backend issues such as multiple accounts with the same credentials.	 *Email and username is already taken, and confirm password invalid notification.*
Back to login button, allowing the user to return to the login screen.	This feature lets the user navigate between pages of the program without having to restart it.	 *Back to login button*

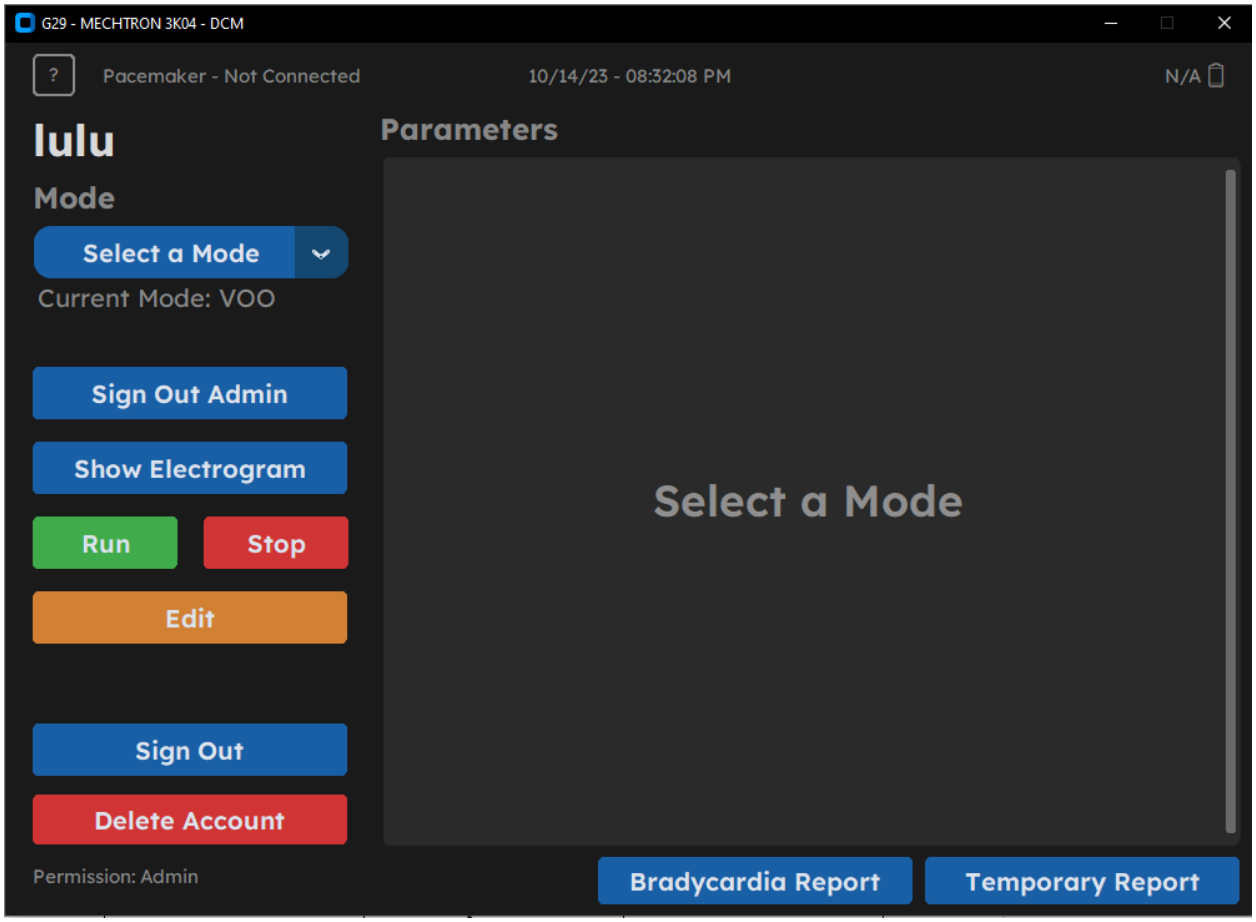
2.2.4.3 - Main Interface Design Choices



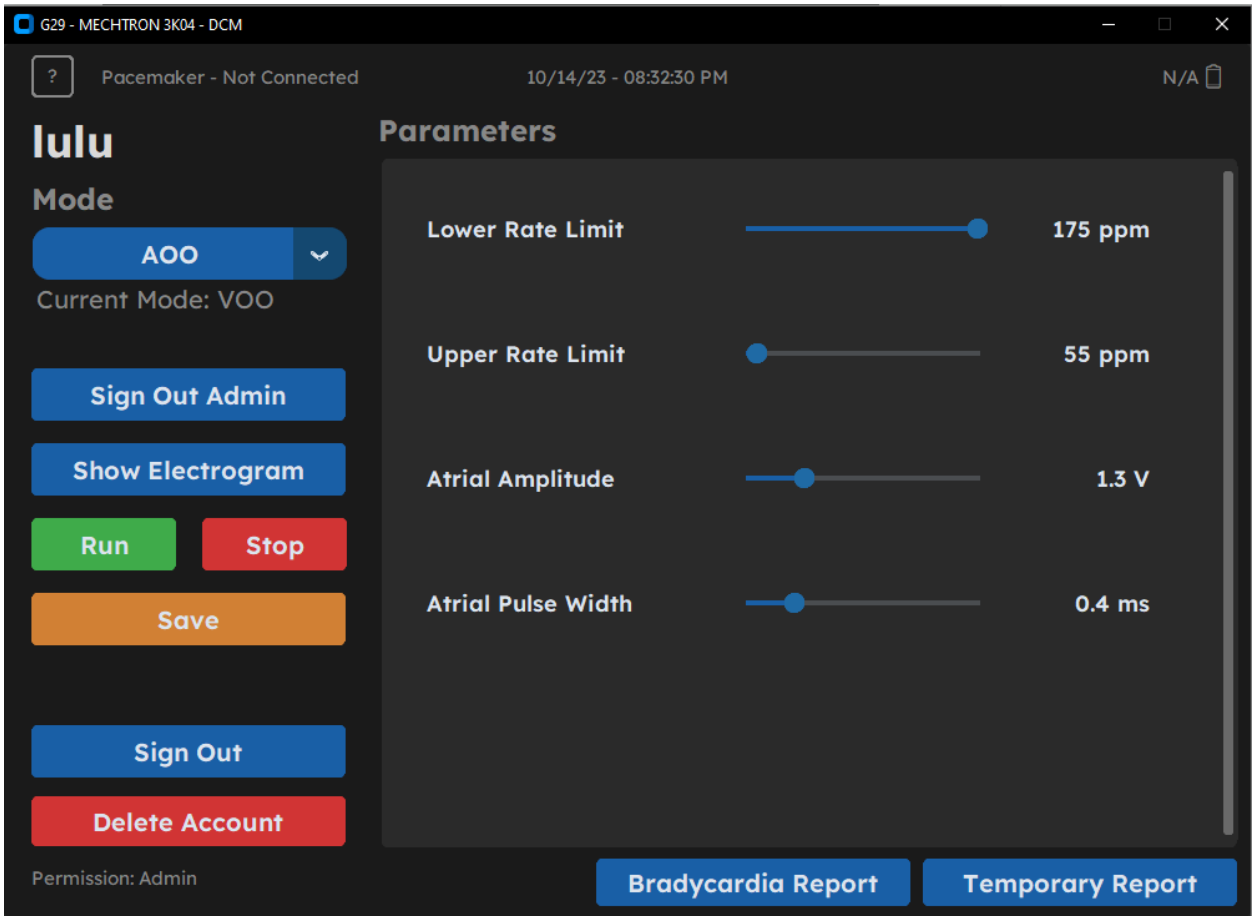
Main interface - client view



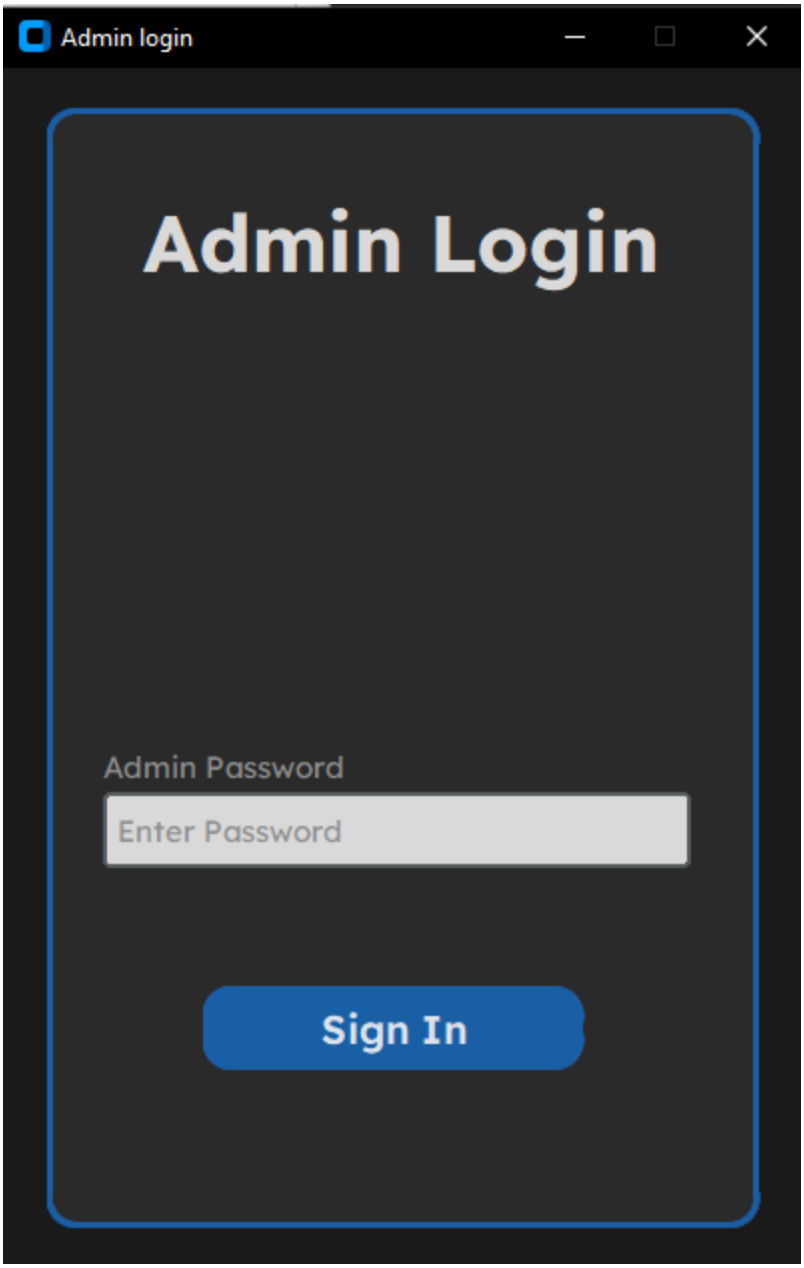
Main interface - client view - AOO selected



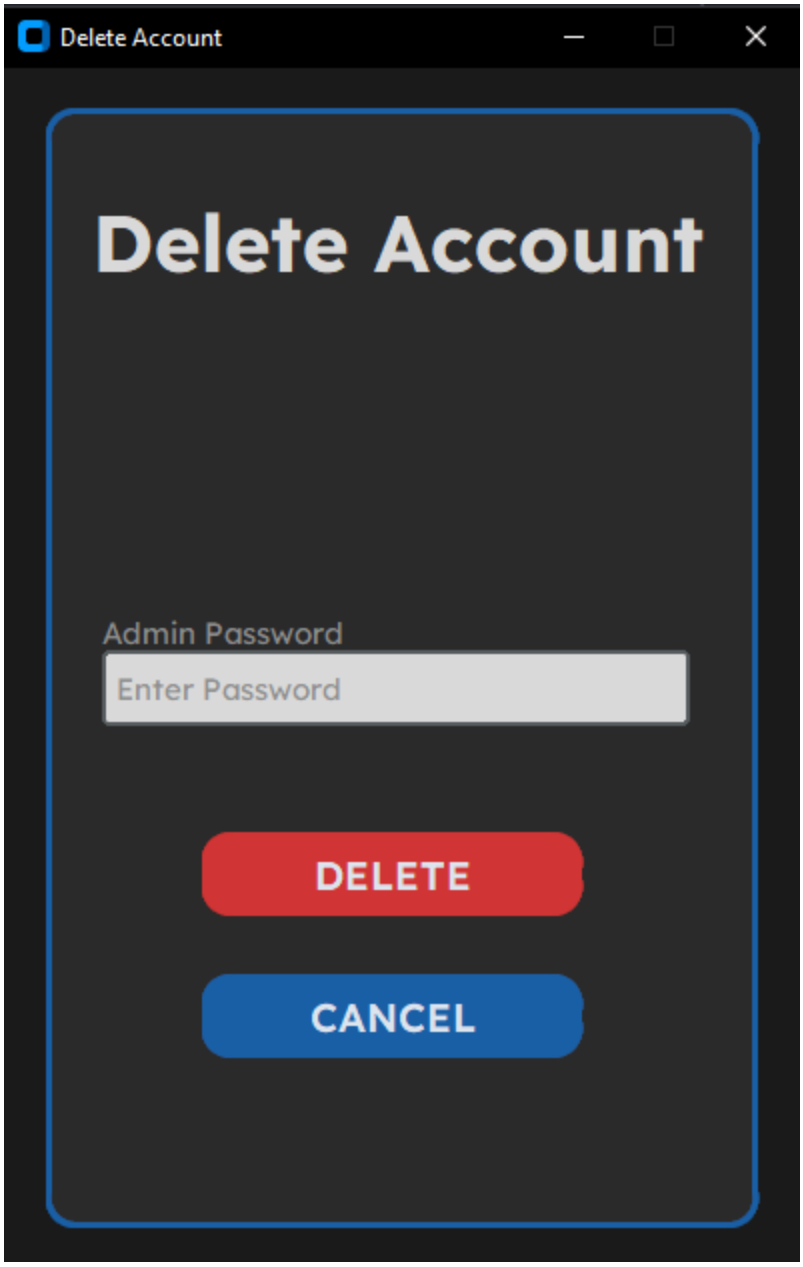
Main Interface - Admin View



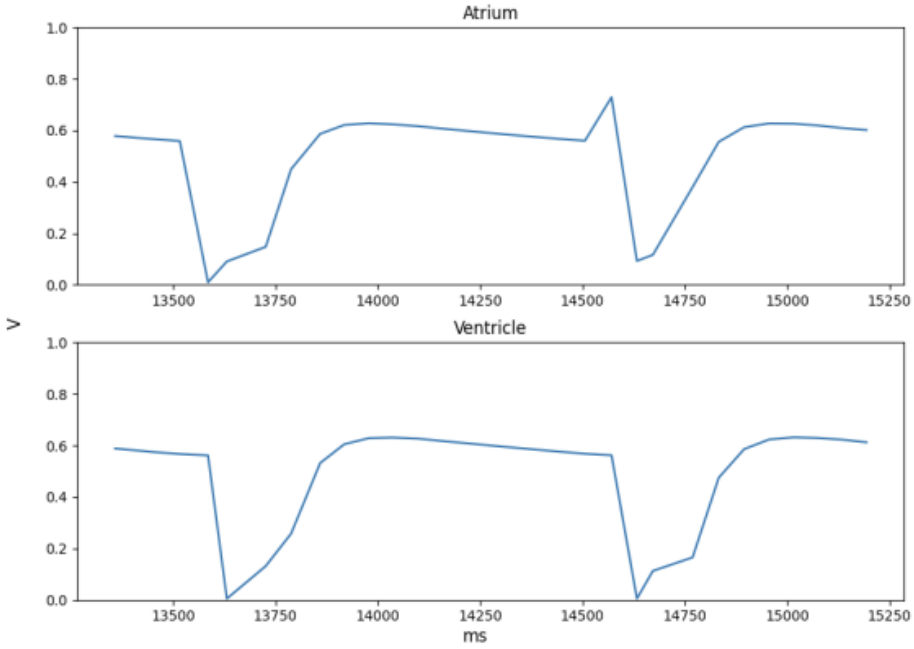
Main Interface - Admin View - AOO selected and modifiable



Admin login pop-up window

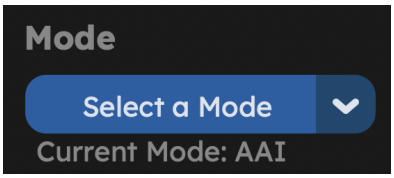
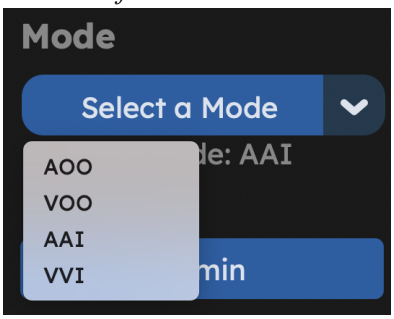
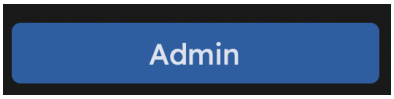


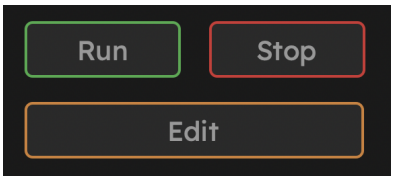
Delete account pop-up window

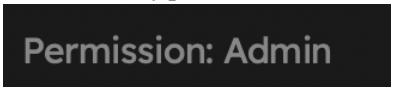
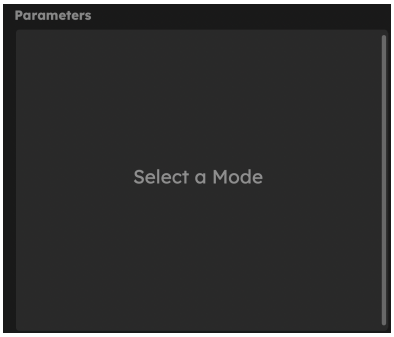
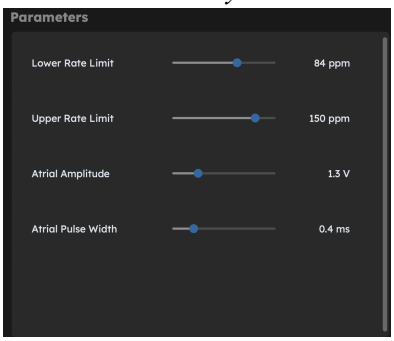
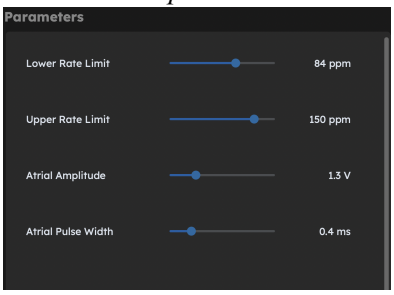


Electrogram graph window

Features added to the main interface along with an explanation and visual aid.

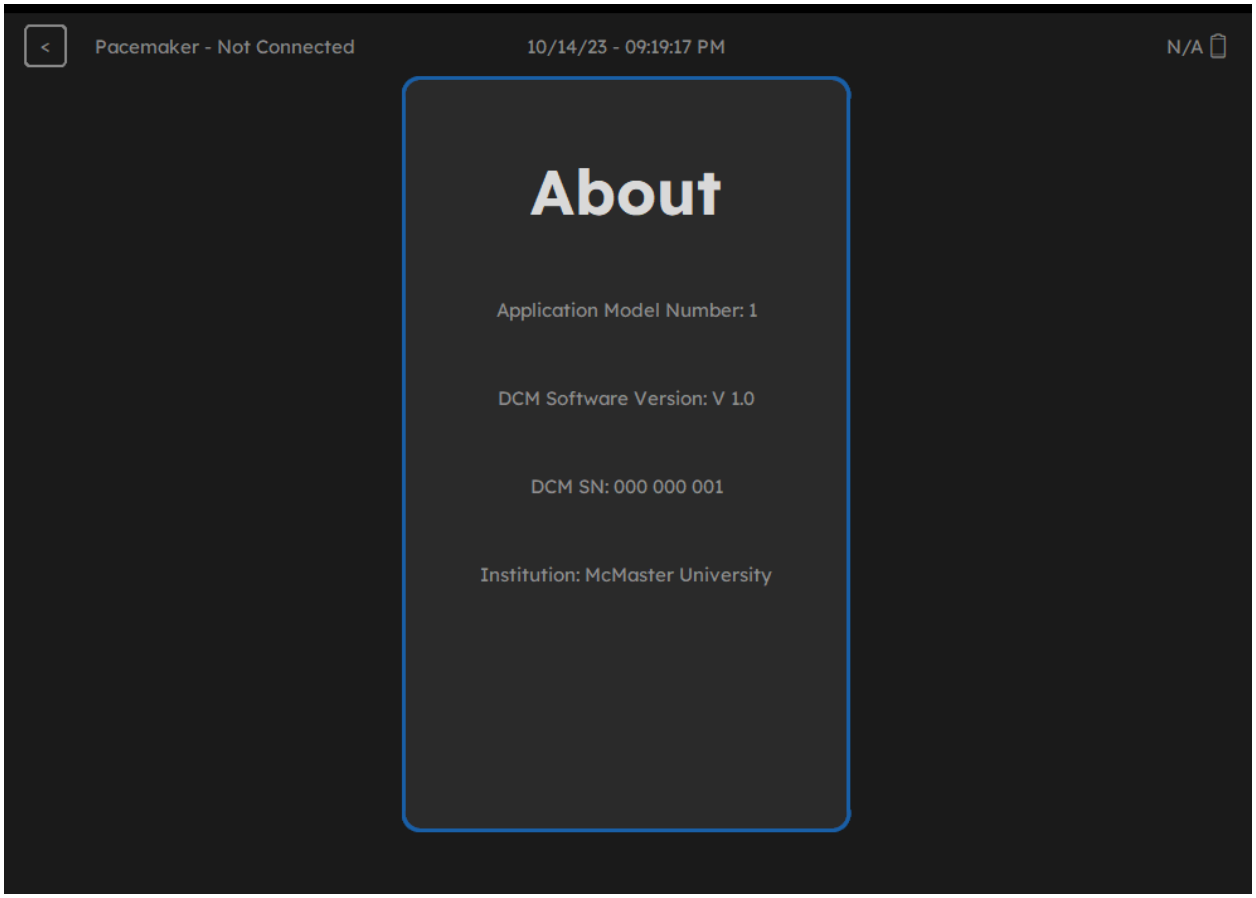
Features Added	Explanation	Visual Aid
Drop-down menu to select the appropriate mode	This allows the user to clearly see all of the modes that are available, making it more user friendly and easy to use. By clicking on the desired mode, the user can then adjust and alter the settings associated with their desired mode.	 *Collapsed version of drop-down menu for mode selection*  *Expanded version of drop-down menu for mode selection.*
Button to log into “Admin” and unlock the Admin privileges	This allows the Admin to control their clients’ settings for their pacemaker. The client cannot alter their own pacemaker settings without the admin to improve safety by not	 *Button to log into admin credentials and gain admin privileges.*

	allowing the client to alter their medical device without the admin (physician, etc.) who has the appropriate background education and experience in this area.	
Button to display the electrogram.	This allows the user to visualize real time electrogram signals coming from the device itself. Both the admin and clients should be able to view this feature at will as this does not disrupt the functionality of the pacemaker, rather it only reads data from it.	 *Button to display the electrogram.*
Buttons to run, stop, and edit the setting for the modes. Buttons are disabled on the client view that must be accessed by entering the right admin password	This make the device safer to use as the client (who does not have the education or experience to alter the settings on their pacemaker) is unable to make changes. Only the admin, physician or a person with the appropriate credentials, is able to make these changes.	 *Run, Stop, and Edit buttons disabled on client view.*  *Run, Stop, and Edit buttons on admin view (not disabled).*
Button to sign out of the account that is currently logged in	Large sign out button that is easily visible and accessible for user to sign out of the account they are currently logged into	 *Button that allows user to sign out of current account that is logged in.*
Buttons are disabled on the client view that must be accessed by entering the right admin password	This only allows the admin to delete an account of a user. This ensures safety of the client so that their pacemaker account is not accidentally deleted by an unqualified person.	 *Button that allows admin to delete account that is disabled on client view.*  *Button to delete account on admin view where button is not*

		disabled
Button that displays the permission status that the user currently has.	This showcases the permission status of the current user so that they are aware of the privileges that they currently have. The permission status will change from client to admin when user logs into admin.	 *Client view of permission status.*  *Admin view of the permission status.*
Side of the screen that displays the parameters and allows the admin to alter and change the mode parameters.	This allows the user to easily see where to modify the parameters (when in admin) and to view the current parameters (when in client view). The sliders allow the admin to adjust the values of the parameters and the upper and lower limits cannot be surpassed to avoid any errors resulting from typos (if it where a textbox where numbers are entered through keyboard).	 *Parameters display when no mode is currently selected.*  *Client view of parameters of AOO where user is unable to edit the parameters.*  *Admin view of parameters of AOO where user is able to edit the parameters.*

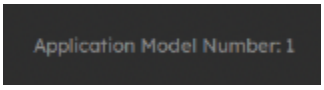
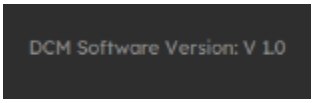
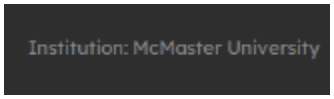
Button that displays the bradycardia report.	The user should be able to download a bradycardia report at will regardless of if they have admin or client privileges. This button was made to be big as there is a lot of information that can be downloaded that may be valuable to the client or the physician.	 *Button to display Bradycardia Report.*
Button that displays the temporary report.	The user should be able to download a temporary report at will regardless of if they they have admin or client privileges. This button was made to be big as there is a lot of information that can be downloaded that may be valuable to the client or the physician.	 *Button to display Temporary Report.*
Username that is displayed near the top of the screen.	The username is displayed in big bold letters so that it is easily visible so the user knows whose account is logged in currently.	 *Image of the username displayed on the main interface.*
Button that saves the parameter settings when modified.	This button is only visible through the admin view after clicking the “Edit” button to edit the parameters. This allows the admin to save the parameters after modifying them so that they are able to adjust the parameters without them being automatically saved and modified.	 *Button to save the parameters that were recently edited.*

2.2.4.4 - About Screen Design Choices



About screen

About Screen Design Features

Features Added	Explanation	Visual Aid
Application model number	Allows the user to know what the application model number of the software is	 *Application model number.*
DCM Software Version	Displayed the current version of the software that the user is currently running	 *DCM software version.*
DCM Serial Number	Allows the user to see what the serial number of their own installed DCM application	 *DCM SN.*
Institution	Displays the institution to the user of where the DCM was created and where its being used.	 *Institution.*

Back button	Back button placed in lieu of the about page button in the top left corner of the application as it is more convenient for the user to return back to their current screen if they wanted to exit out of the about page.	 *Back button.*

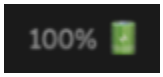
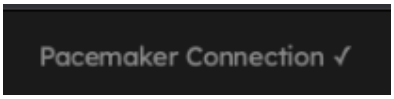
2.3 - Future Pacemaker Design Changes

For future improvements, a major efficiency upgrade could come forth in the way of improving the cohesion and interaction between the accelerometer data subsystem and the serial communication system workflow. In the current model, when these 2 feature are running simultaneously, the program is not as efficient as compared to when one is off. A way to improve this would be to implement the serial communication function and the pacemaker function as two parallel states. This will improve the model throughput and allow the program to function in a more power efficient manner as well as improving functionality and speed.

2.4 - Future DCM Design Changes

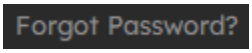
2.4.1 - General Future Design Changes

General design changes of the DCM in for the future

Feature	Future Changes	Visual Aid
Battery status visual prompt.	Successfully connect the pacemaker to the DCM program and integrate the battery status information into the visual prompt. Currently, battery is displayed as 100%, which is not a real measurement of the battery percentage.	 *Battery status visual prompt.*
Identify a pacemaker with a different serial number when plugged into DCM	Currently, the DCM is only able to detect if a pacemaker is plugged in or not. In the future, it would be ideal if the DCM is able to differentiate between different pacemakers using their serial number.	 *Pacemaker connection status when any pacemaker is plugged in*

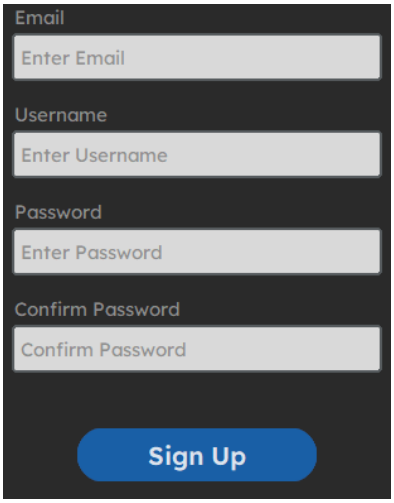
2.4.2 - Login Screen Future Design Changes

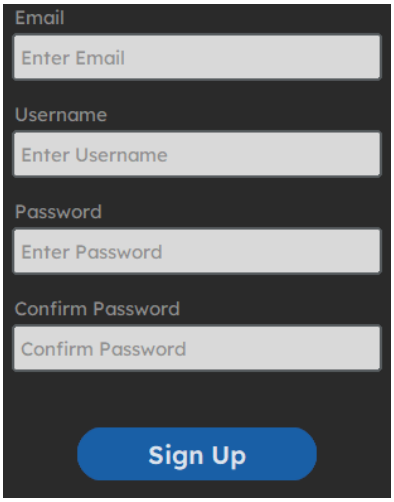
Login screen design changes of the DCM in for the future

Feature	Future Changes	Visual Aid
Forgot password button, allowing a user the option to reset their password.	Send an email to a user-specific email address when a user submits a form including their username to renew their password.	 *Forgot password button*

2.4.3 - Register Screen Future Design Changes

Register screen design changes of the DCM in for the future

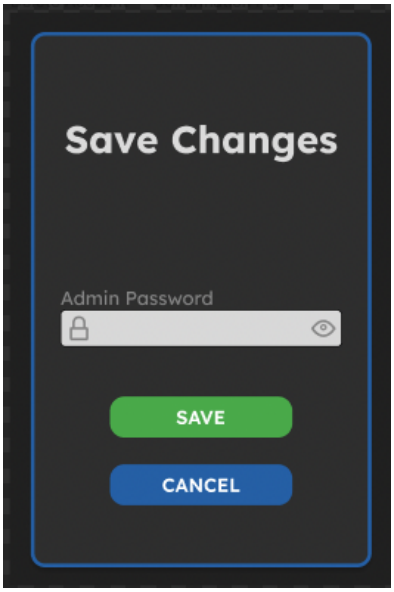
Feature	Future Changes	Visual Aid
Additional feature to the email textbox to improve user experience.	Send an email to a user specific email address when the user makes an account as a "Welcome" message.	 *Sign up screen where user enters their information to create an account.*

Additional feature to the email address textbox to confirm and verify that the correct information is entered when signing up for an account.	Check if the email address entered is a valid email address before allowing user to create an account. Ideally, a “warning” message in red text will appear if the email address entered is not valid.	 *Current sign up screen, but with the added feature, if the email address entered is not valid then a “warning” message will appear in red text near the “Email” textbox that prompts user to enter a valid email address.*

2.4.4 - Main Interface Future Design Changes

Main interface design changes of the DCM in for the future

Feature	Future Changes	Visual Aid
Bradycardia Report and Temporary Report	Use the data collected to display actual information instead of having buttons without any function as this feature is currently not functional for Assignment 2. In the future, a plotting library in Python will most likely be utilized to aid in plotting real time readings from the pacemaker for the data.	 *Buttons for the reports that have not been implemented yet*

Save changes confirmation	When saving the parameters for a mode, a prompt should appear and the admin password should be entered to confirm a parameter change. This will prevent accidental changes.	 *Save changes button*

3.0 - Test Cases

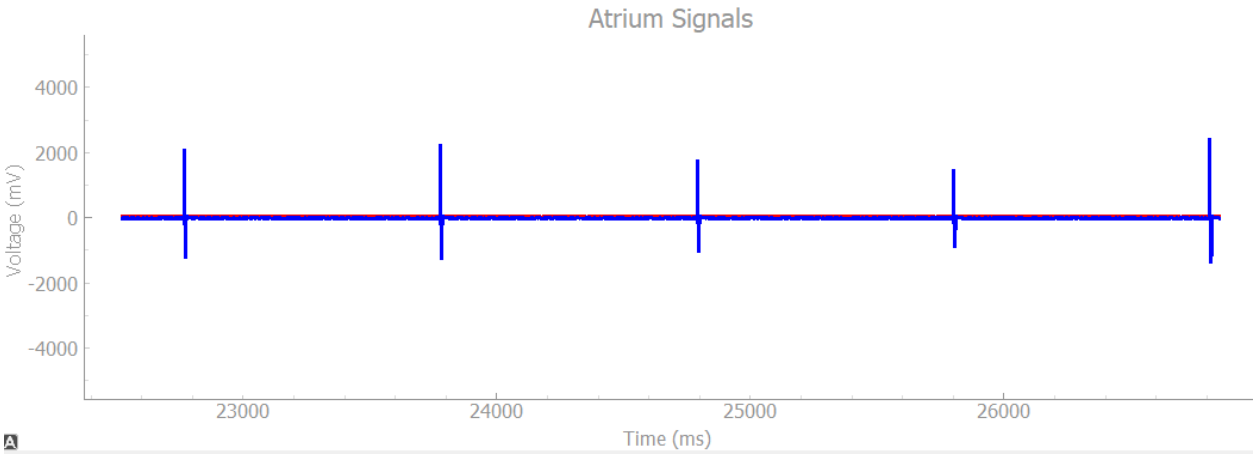
3.1 - Pacemaker Test Cases

The following tables represent test cases for the newly added rate adaptive pacemaker modes.

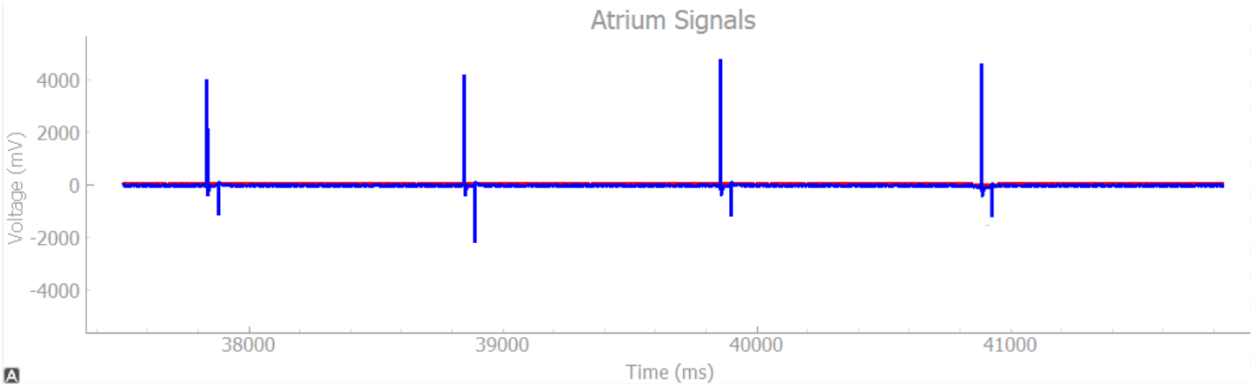
To verify that the pulse signals sent by the pacemaker are at the adequate time interval, it is important to check if adjusting the pulse width and amplitude corresponds to a change in the pacing behaviour.

Pulse width and amplitude test case 1

Pacemaker Settings	**Heartview Settings**
Mode: AOOR	Natural Atrium: Off
Pulse Width: 3 ms then 20 ms	Natural Ventricle: Off
Pulse Amp: 1.5 V then 3.9 V	Natural Heart Rate: 30 bpm
Lower Rate: 60 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker will send a higher amplitude for the pulse in the second test and the pulse width will be wider	
Actual Output: Result is as expected	
Result	PASS



Heartview output for pulse width and amplitude testing, setting 1

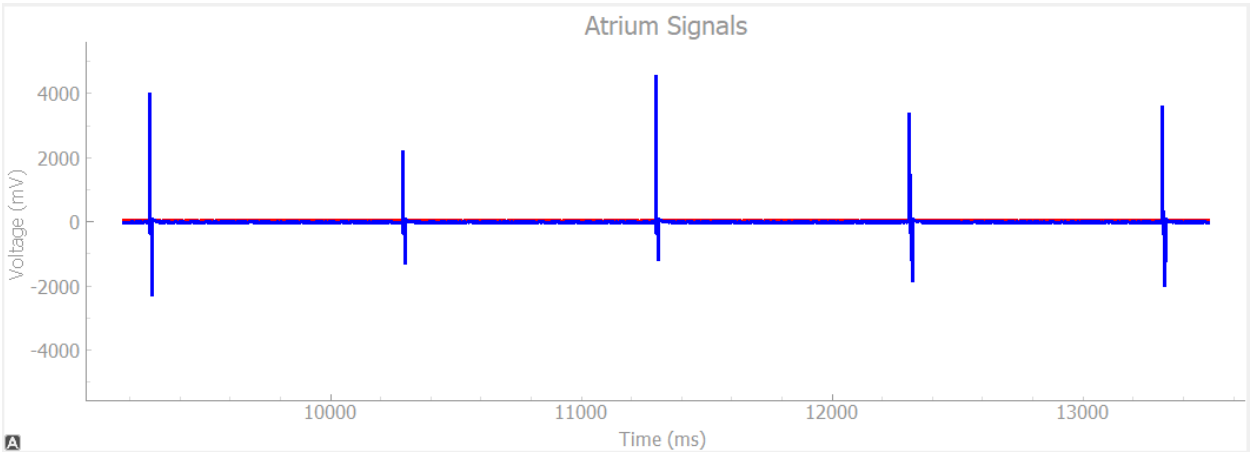


Heartview output for pulse width and amplitude testing, setting 2

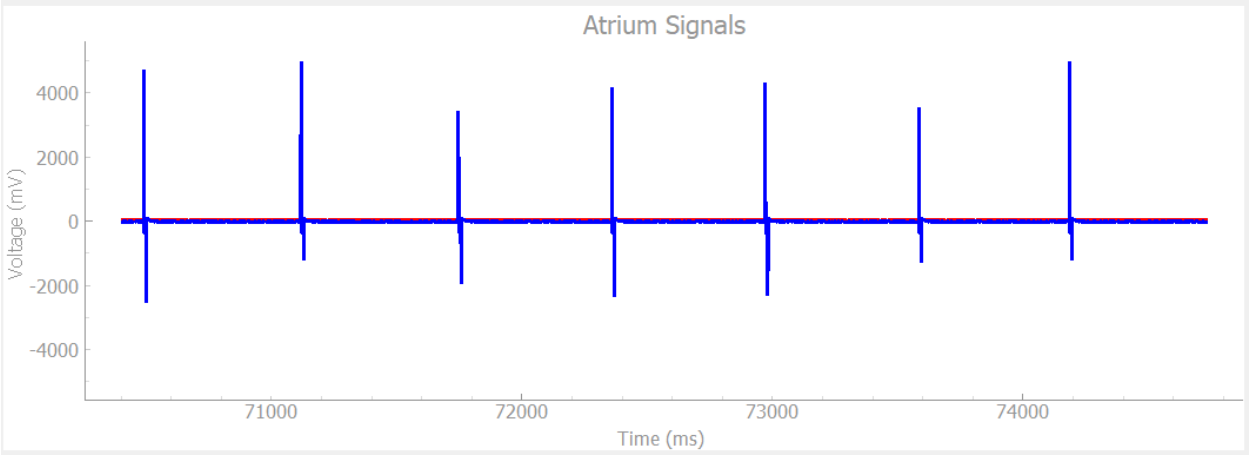
To verify the adaptive pacing modes, AOOR and VOOR, correctly, 2 tests must be done at minimum. The first test is to introduce activity and then confirm that the pacing rate gradually increases. The second is to introduce natural heart beats and observe the change of pacing while the board is shaken.

AOOR test case 1

Pacemaker Settings	**Heartview Settings**
Mode: AOOR	Natural Atrium: Off
Pulse Width: 20 ms	Natural Ventricle: Off
Pulse Amp: 5 V	Natural Heart Rate: 30 bpm
Lower Rate: 60 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker will make the pacing rate faster when board is shaken	
Actual Output: Result is as expected	
Result	PASS



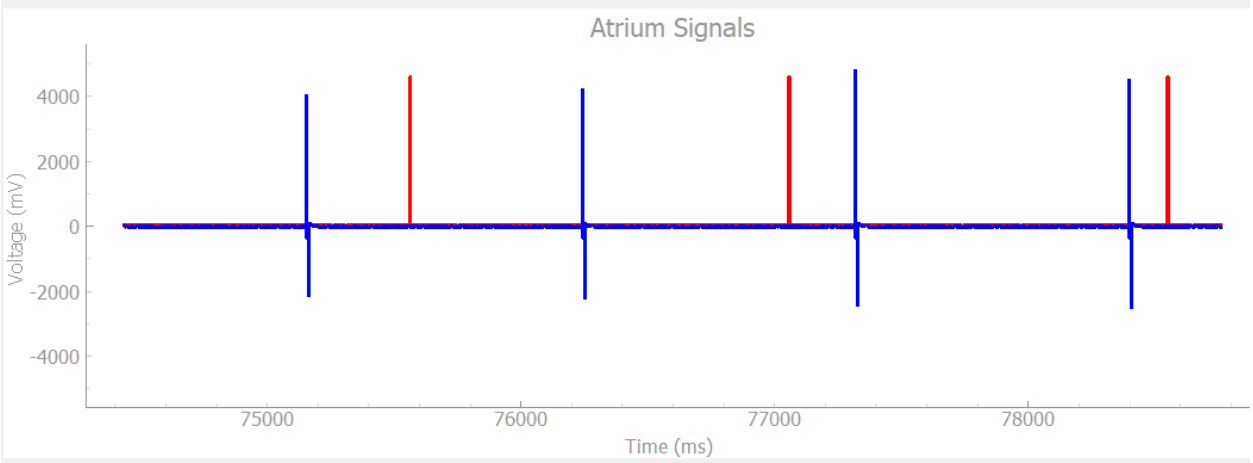
Heartview output for AOOR test case 1, resting pace



Heartview output for AOOR test case 1, after activity introduced

AOOR test case 2

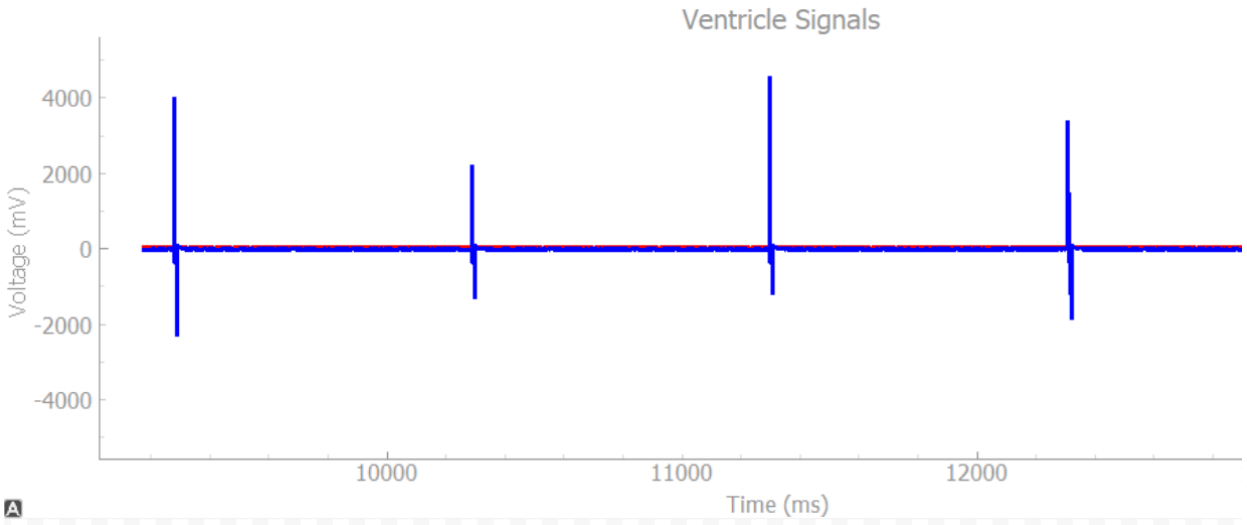
Pacemaker Settings	**Heartview Settings**
Mode: AOOR	Natural Atrium: On
Pulse Width: 20 ms	Natural Ventricle: Off
Pulse Amp: 5 V	Natural Heart Rate: 30 bpm
Lower Rate: 60 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker will make the pacing rate faster when board is shaken without regard for the natural beats.	
Actual Output: Result is as expected	
Result	PASS



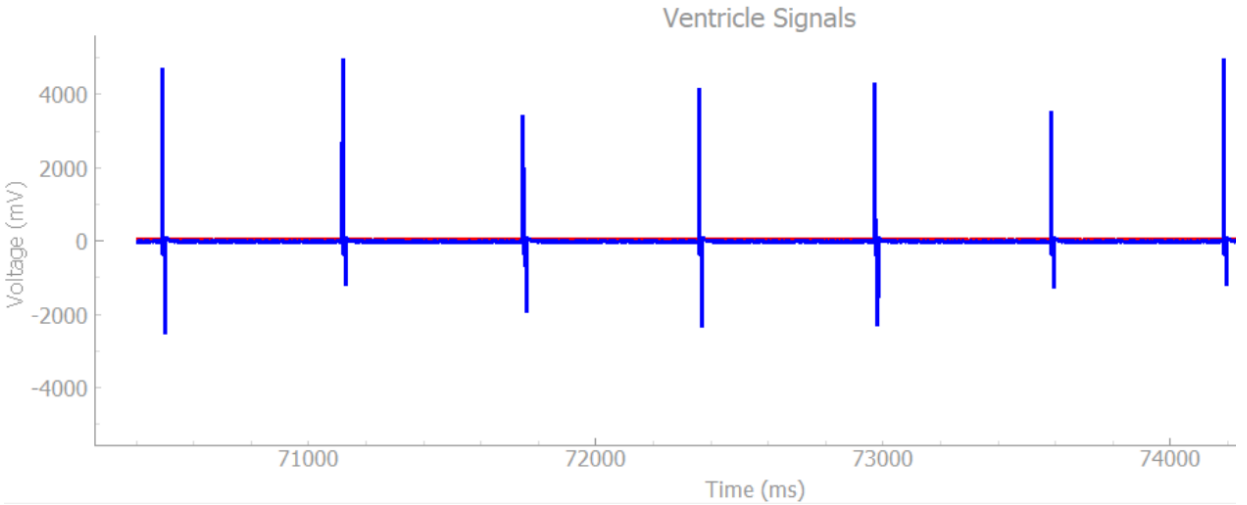
Heartview output for AOOR test case 2

VOOR test case 1

Pacemaker Settings	**Heartview Settings**
Mode: VOOR	Natural Atrium: Off
Pulse Width: 20 ms	Natural Ventricle: Off
Pulse Amp: 5 V	Natural Heart Rate: 30 bpm
Lower Rate: 20 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker will make the pacing rate faster when board is shaken	
Actual Output: Result is as expected	
Result	PASS



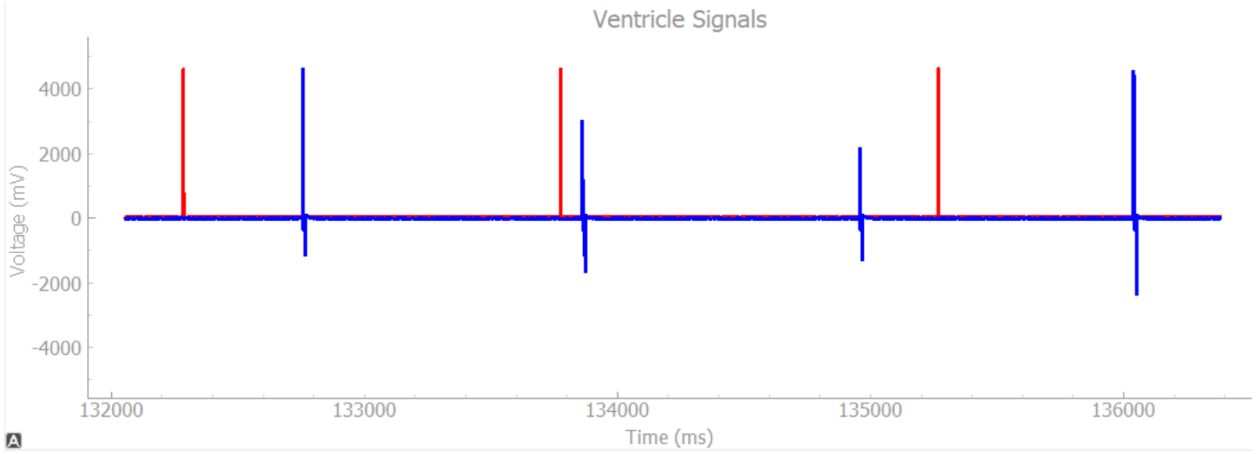
Heartview output for VOOR test case 1, resting pace



Heartview output for VOOR test case 1, after activity introduced

VOOR test case 2

Pacemaker Settings	**Heartview Settings**
Mode: VOOR	Natural Atrium: Off
Pulse Width: 20 ms	Natural Ventricle: On
Pulse Amp: 5 V	Natural Heart Rate: 45 bpm
Lower Rate: 20 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker will make the pacing rate faster when board is shaken without regard for the natural beats.	
Actual Output: Result is as expected	
Result	PASS

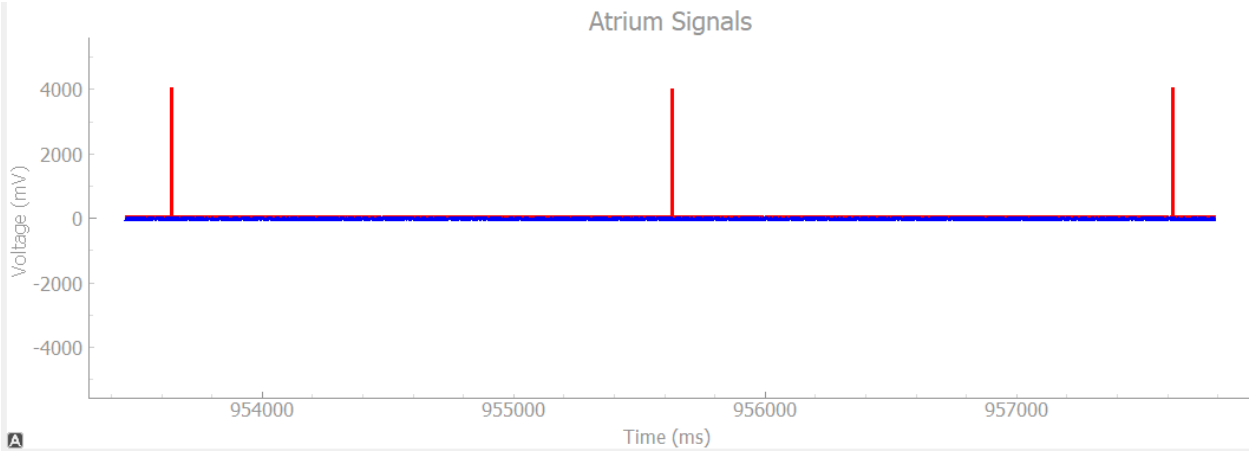


Heartview output for VOOR test case 2

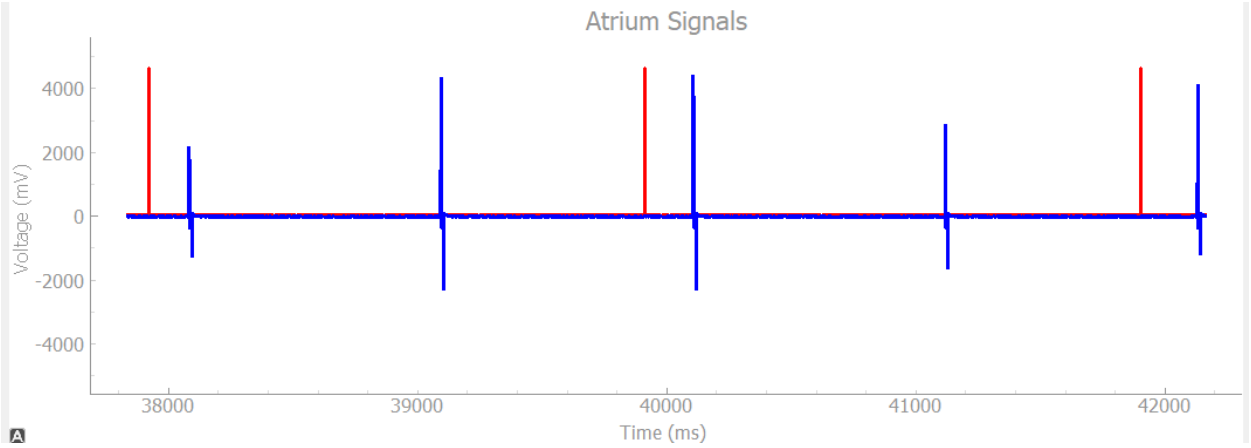
For AAIR and VVIR, again 2 tests are needed. First test is to check if the pacemaker will NOT pace if the Lower Rate and Natural Heart Rate are the same, however, when the board is shaken, it should take over and more paces should be observed. The second test is to set the pacemaker Lower Rate higher than the Natural Heart Rate to verify if it will send paces. Then when the pacemaker is shaken, it should see even more paces.

AAIR test case 1

Pacemaker Settings	**Heartview Settings**
Mode: AAIR	Natural Atrium: On
Pulse Width: 20 ms	Natural Ventricle: Off
Pulse Amp: 5 V	Natural Heart Rate: 30 bpm
Lower Rate: 30 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker should not pace with no activity but then should pace when board is shaken.	
Actual Output: Result is as expected.	
Result	PASS



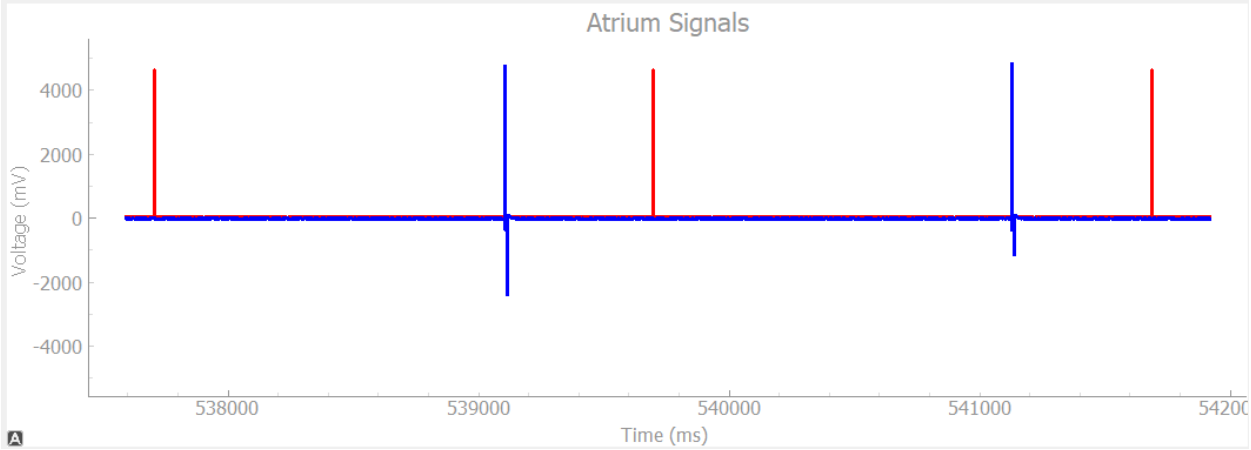
Heartview output for AAIR test case 1, resting pace



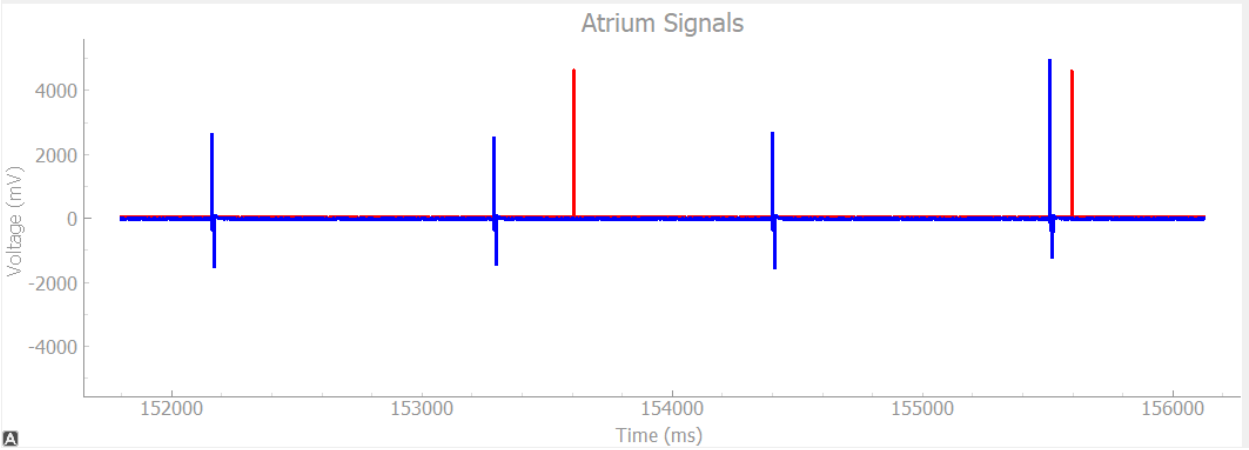
Heartview output for AAIR test case 1, after activity introduced

AAIR test case 2

Pacemaker Settings	**Heartview Settings**
Mode: AAIR	Natural Atrium: On
Pulse Width: 20 ms	Natural Ventricle: Off
Pulse Amp: 5 V	Natural Heart Rate: 30 bpm
Lower Rate: 60 ppm	Natural AV Delay: 30 ms
Expected Output: Pacemaker should pulse at rest since lower rate is higher than natural and then should increase pulsing rate as activity is introduced.	
Actual Output: Result is as expected	
Result	PASS



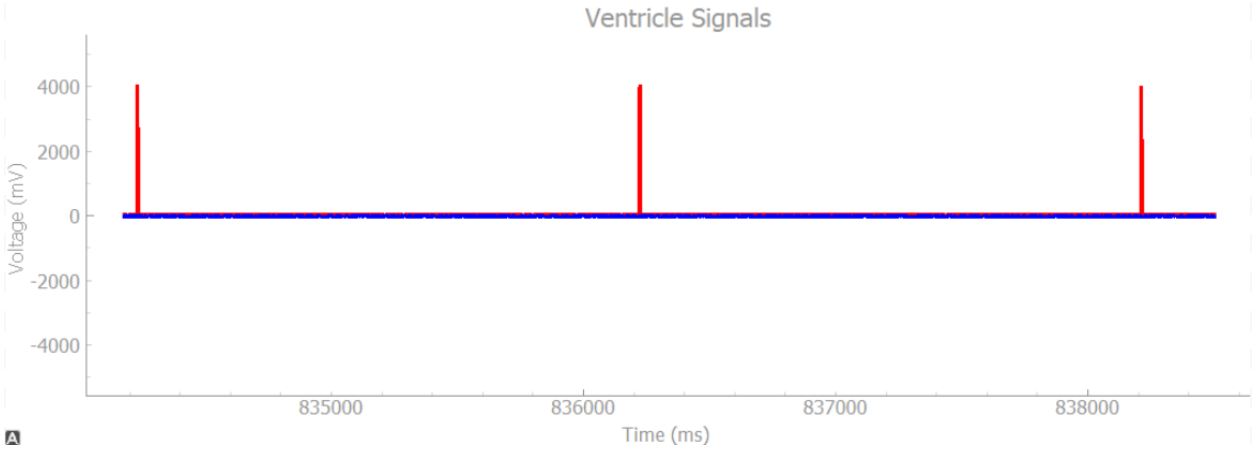
Heartview output for AAIR test case 2, resting pace



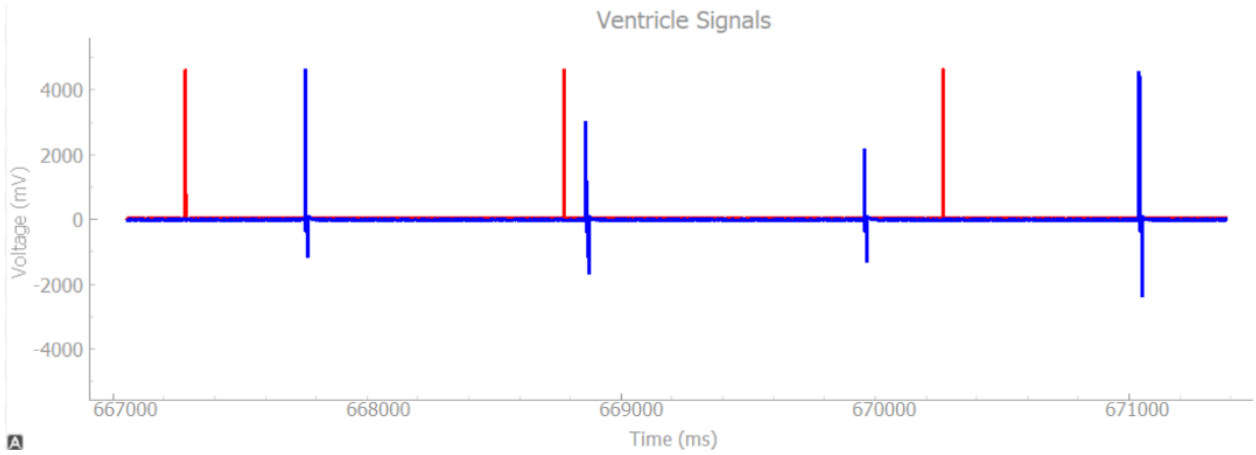
Heartview output for AAIR test case 2, after activity introduced

VVIR test case 1

Pacemaker Settings	**Heartview Settings**
Mode: VVIR	Natural Atrium: Off
Pulse Width: 20 ms	Natural Ventricle: On
Pulse Amp: 5 V	Natural Heart Rate: 30 bpm
Lower Rate: 30 ppm	Natural AV Delay: 30 ms
Expected Output: The pacemaker should not pace with no activity but then should pace when board is shaken.	
Actual Output: Result is as expected	
Result	PASS



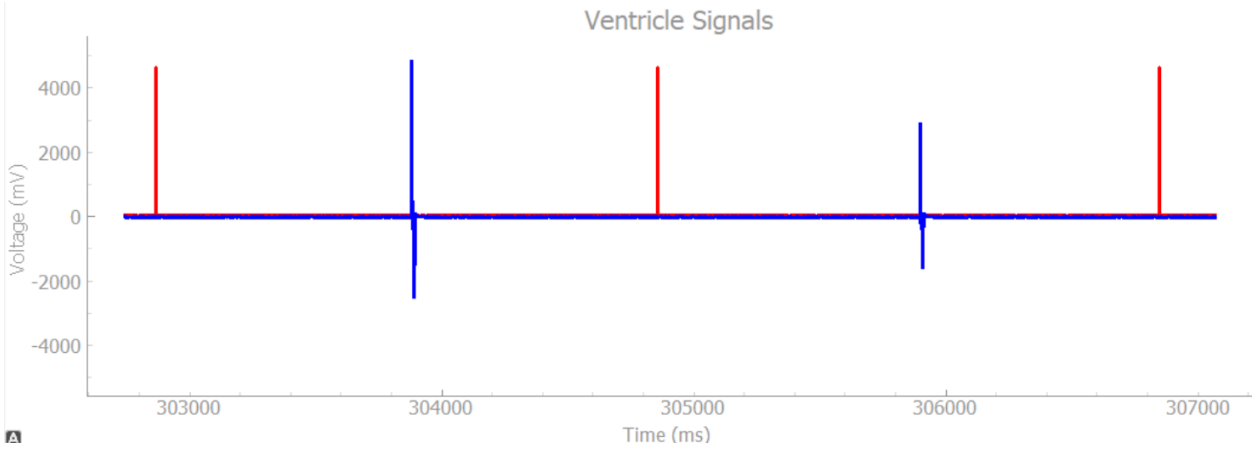
Heartview output for VVIR test case 1, resting pace



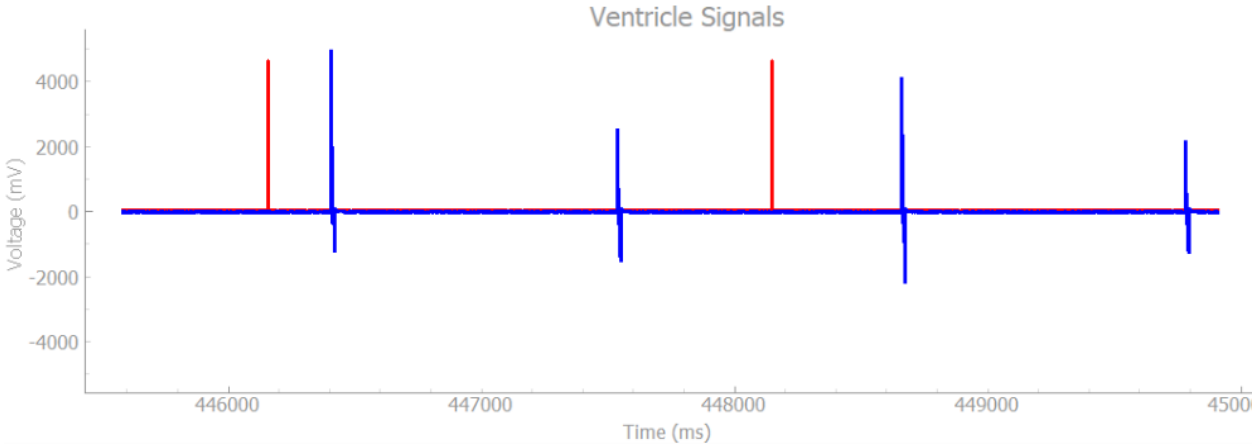
Heartview output for VVIR test case 1, activity introduced

VVIR test case 2

Pacemaker Settings	**Heartview Settings**
Mode: VOOR	Natural Atrium: Off
Pulse Width: 20 ms	Natural Ventricle: On
Pulse Amp: 5 V	Natural Heart Rate: 30bpm
Lower Rate: 60 ppm	Natural AV Delay: 30 ms
Expected Output: Pacemaker should pulse at rest since lower rate is higher than natural and then should increase pulsing rate as activity is introduced.	
Actual Output: Result is as expected	
Result	PASS



Heartview output for VVIR test case 2, resting pace



Heartview output for VVIR test case 2, after activity introduced

3.2 - DCM Test Cases

The following tables represent test cases for newly added modules/functionalities to the DCM front end and back end as well as pacemaker connection.

3.2.1 - Global Functions

Format_data(current_mode_data : dict):

Test to see if the formatting data works as intended. The data to be formatted is the format that the pacemaker accepts data in.

Test cases with the expected result, actual result, and pass/fail status

id	Test Case	Expected Result	Actual Result	Pass/Fail
1	{ "Lower Rate Limit": 120, "Upper Rate Limit": 120, "Atrial Amplitude": 5.0, "Atrial Pulse Width": 30 }	[0, 50, 14, 0, 0, 0, 0, 50, 0, 29, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[0, 50, 14, 0, 0, 0, 0, 50, 0, 29, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Pass
2	{}	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Pass
3	{ "Lower Rate Limit": 60, "Upper Rate Limit": 120, "Maximum Sensor Rate": 120, "Fixed AV Delay": 150, "Dynamic AV Delay": "Off", "Sensed AV Delay Offset": "Off", "Atrial Amplitude": 5, "Ventricular Amplitude": 5, "Atrial Pulse Width": 1,	[0, 14, 14, 14, 8, 0, 0, 50, 50, 0, 0, 25, 25, 17, 10, 10, 0, 0, 0, 1, 0, 0, 3, 2, 7, 3]	[0, 14, 14, 14, 8, 0, 0, 50, 50, 0, 0, 25, 25, 17, 10, 10, 0, 0, 0, 1, 0, 0, 3, 2, 7, 3]	Pass

	"Ventricular Pulse Width": 1, "Atrial Sensitivity": 2.5, "Ventricular Sensitivity": 2.5, "VRP": 320, "ARP": 250, "PVARP": 250, "PVARP Extension": "Off", "Hysteresis": "Off", "Rate Smoothing": "Off", "ATR Duration": 20, "ATR Fallback Mode": "Off", "ATR Fallback Time": 1, "Activity Threshold": "Med", "Reaction Time": 30, "Response Factor": 8, "Recovery Time": 5 }			
4	0	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Error	Fail
5	“Off”	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Error	Fail
6	{"ok" : 1}	Error	Error	Pass

Encrypt_password(password : str):

Test to see if the password encryption works on a variety of different passwords.

Test cases with the expected result, actual result, and pass/fail status

id	Test Case	Expected Result	Actual Result	Pass/Fail

1	"password"	“p~a~s~s~w~o~r~d~”	“p~a~s~s~w~o~r~d~”	Pass
2	“12345”	“1~2~3~4~5~”	“1~2~3~4~5~”	Pass
3	12345	Error	Error	Pass
4	“password123!@”	“p~a~s~s~w~o~r~d~1~2~3~!~@~”	“p~a~s~s~w~o~r~d~1~2~3~!~@~”	Pass
5	“”	“~”	“”	fail
	“ ”	“ ~”	“ ~”	Pass

decrypt_password(encrypted_password : str):

Test to decrypt encrypted passwords to see if any password a user may use to sign up with is valid.

Test cases with the expected result, actual result, and pass/fail status

id	Test Case	Expected Result	Actual Result	Pass/Fail
1	“p~a~s~s~w~o~r~d~”	“password”	“password”	Pass
2	“ ~“	“ “	“ “	Pass
3	“~”	“~”	“~”	Pass
4	“HEI1L1L1O1”	“HELLO”	“HELLO”	Pass
5	“”	“”	“”	Pass
6	“p~a~s~s~w~o~r~d~1~2~3~!~@~”	“password123!@”	“password123!@”	Pass

3.2.2 - Main App Module Methods and Functionality

Pacemaker Connection

Test to see different potential states the DCM may enter based on different variables such as pacemaker connection, admin permissions, etc.

Test cases with the expected result, actual result, and pass/fail status

id	Test state	Expected Result	Actual Result	Pass/Fail

1	Login Screen - Pacemaker connected	Display pacemaker connection, don't send parameters	Display pacemaker connection, don't send parameters	Pass
2	Sign up Screen - Pacemaker connected	Display pacemaker connection, don't send parameters	Display pacemaker connection, don't send parameters	Pass
3	Main Menu - Pacemaker connected - current mode paced : "Off"	Display Pacemaker connection, don't send parameters	Display Pacemaker connection, don't send parameters	Pass
4	Main Menu - Pacemaker connected - current mode paced : "AOO", perms = client	Display Pacemaker connection, send parameters for AOO	Display Pacemaker connection, send parameters for AOO	Pass
5	Main Menu - Pacemaker connected - current mode paced : "AOO", perms = Admin	Display Pacemaker connection, send parameters for AOO, activate Run, Stop, Verify, Electrogram buttons	Display Pacemaker connection, send parameters for AOO, activate Run, Stop, Verify, Electrogram buttons	Pass

Pacemaker Disconnection

Test to see different potential states the DCM may enter based on different variables such as a lack of pacemaker connection, admin permissions, etc.

Test cases with the expected result, actual result, and pass/fail status for pacemaker disconnection

id	**Test state**	**Expected Result**	**Actual Result**	**Pass/Fail**
1	Login Screen - Pacemaker Disconnected	Display pacemaker disconnection	Display pacemaker disconnection	Pass
2	Sign up Screen - Pacemaker	Display	Display	Pass

	Disconnected	pacemaker disconnection	pacemaker disconnection	
3	Main Menu - Pacemaker Disconnected - current mode paced : “Off”	Display pacemaker disconnection, turn off button functionality for run, stop, verify and egram	Display pacemaker disconnection, turn off button functionality for run, stop, verify and egram	Pass
4	Main Menu - Pacemaker Disconnected - current mode paced : “AOO”, perms = client	Display pacemaker disconnection, turn off button functionality for run, stop, verify and egram	Display pacemaker disconnection, turn off button functionality for run, stop, verify and egram	Pass
5	Main Menu - Pacemaker Disconnected - current mode paced : “AOO”, perms = Admin	Display pacemaker disconnection, turn off button functionality for run, stop, verify and egram	Display pacemaker disconnection, turn off button functionality for run, stop, verify and egram	Pass

_start_button_cmd(self, selected_mode : str):

Test to see different the behaviour of pressing the start button based on the state of the DCM.

Test cases with the expected result, actual result, and pass/fail status for the start button

id	Test state	Expected Result	Actual Result	Pass/Fail
1	Mode Selected : “Off”, Current Selected Mode : “Off”, Pacemaker : Disconnected	Can’t Press	Can’t Press	Pass
2	Mode Selected : “AOO”, Current Selected Mode : “Off”, Pacemaker : Disconnected	Can’t Press	Can’t Press	Pass
3	Mode Selected : “Off”, Current Selected Mode : “Off”, Pacemaker : Connected	Do Nothing	Do Nothing	Pass
4	Mode Selected : “AOO”, Current Selected Mode : “Off”,	Pace AOO with parameters	Pace AOO with parameters	Pass

	Pacemaker : Connected			
5	Mode Selected : “AAIR”, Current Selected Mode : “AOO”, Pacemaker : Connected	Pace AIIR with parameters	Pace AIIR with parameters	Pass
6	Mode Selected : “AOO”, Current Selected Mode : “AOO”, Pacemaker : Connected	Pace AOO with parameters	Pace AOO with parameters	Pass

_stop_button_cmd(self):

Test to see different the behaviour of pressing the stop button based on the state of the DCM.

Test cases with the expected result, actual result, and pass/fail status for the stop button

id	Test state	Expected Result	Actual Result	Pass/Fail
1	Mode Selected : “Off”, Current Selected Mode : “Off”, Pacemaker : Disconnected	Can’t Press	Can’t Press	Pass
2	Mode Selected : “AOO”, Current Selected Mode : “Off”, Pacemaker : Disconnected	Can’t Press	Can’t Press	Pass
3	Mode Selected : “Off”, Current Selected Mode : “Off”, Pacemaker : Connected	Do Nothing	Do Nothing	Pass
4	Mode Selected : “AOO”, Current Selected Mode : “Off”, Pacemaker : Connected	Stop Pacing	Stop Pacing	Pass
5	Mode Selected : “AOO”, Current Selected Mode : “AOO”, Pacemaker : Connected	Stop Pacing	Stop Pacing	Pass
6	Mode Selected : “AAIR”, Current Selected Mode : “AAIR”, Pacemaker : Connected	Stop Pacing	Stop Pacing	Pass

3.2.3 - User Module

get_formatted_data(self):

Test to see if the formatted data is returned correctly from a user based on their current saved mode

Test cases with the expected result, actual result, and pass/fail status

id	Test Case	Expected Result	Actual Result	Pass/Fail
1	Self._current_mode = “Off”	[0] * 26	[0] * 26	Pass
2	Self._current_mode = “AOO”, "AOO": { "Lower Rate Limit": 120, "Upper Rate Limit": 120, "Atrial Amplitude": 5.0, "Atrial Pulse Width": 30 }	[1, 50, 14, 0, 0, 0, 0, 50, 0, 29, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 50, 14, 0, 0, 0, 0, 50, 0, 29, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Pass
3	Self._current_mode = “AAIR”, “AAIR” : { "Lower Rate Limit": 60, "Upper Rate Limit": 120, "Maximum Sensor Rate": 120, "Atrial Amplitude": 5, "Atrial Pulse Width": 1, "Atrial Sensitivity": 2.5, "ARP": 250, "PVARP": 250, "Hysteresis": "Off", "Rate Smoothing": "Off", "Activity Threshold": "Med", "Reaction Time": 30, "Response Factor": 8, "Recovery Time": 5 }	[6, 14, 14, 14, 0, 0, 0, 50, 0, 0, 0, 25, 0, 0, 10, 10, 0, 0, 0, 0, 0, 0, 3, 2, 7, 3]	[6, 14, 14, 14, 0, 0, 0, 50, 0, 0, 0, 25, 0, 0, 10, 10, 0, 0, 0, 0, 0, 0, 3, 2, 7, 3]	Pass
4	Self._current_mode = “DDDR”, DDDR": { "Lower Rate Limit": 60, "Upper Rate Limit": 120, "Maximum Sensor Rate": 120, "Fixed AV Delay": 150, "Dynamic AV Delay": "Off", "Sensed AV Delay Offset": "Off", "Atrial Amplitude": 5, "Ventricular Amplitude": 5, "Atrial Pulse Width": 1, "Ventricular Pulse Width":	[10, 14, 14, 14, 8, 0, 0, 50, 50, 0, 0, 25, 25, 17, 10, 10, 0, 0, 0, 1, 0, 0, 3, 2, 7, 3]	[10, 14, 14, 14, 8, 0, 0, 50, 50, 0, 0, 25, 25, 17, 10, 10, 0, 0, 0, 1, 0, 0, 3, 2, 7, 3]	Pass

	1, "Atrial Sensitivity": 2.5, "Ventricular Sensitivity": 2.5, "VRP": 320, "ARP": 250, "PVARP": 250, "PVARP Extension": "Off", "Hysteresis": "Off", "Rate Smoothing": "Off", "ATR Duration": 20, "ATR Fallback Mode": "Off", "ATR Fallback Time": 1, "Activity Threshold": "Med", "Reaction Time": 30, "Response Factor": 8, "Recovery Time": 5 }			

3.2.4 - Scrolling Parameter Module

scroll_parameters_frame(master : customtkinter, current_mode_data : dict, current_mode : str, can_edit : bool, send_data_func : function, init_data_func : function, **kwargs):

Test cases to see the behaviour of the frame containing sliders for parameters based on the given state of the system.

Test cases with the expected result, actual result, and pass/fail status

id	Test Case	Expected Result	Actual Result	Pass/Fail
1	Current_mode_data : None, Current_mode : None, Can_edit : False,	Display Select a mode	Display Select a mode	Pass
2	Current_mode_data : Data for AOO, Current_mode : “AOO”, Can_edit : False,	Display parameter soldiers for AOO. Greyed out sldiers, cannot edit	Display parameter soldiers for AOO. Greyed out sldiers, cannot edit	Pass
3	Current_mode_data : Data for AOO, Current_mode : “AOO”, Can_edit : True,	Display parameter soldiers for AOO. blue sliders to edit	Display parameter soldiers for AOO. blue	Pass

			sliders to edit	
4	Current_mode_data : None, Current_mode : None, Can_edit : True,	Display Select a mode	Display Select a mode	Pass
5	Current_mode_data : Data for AAIR, Current_mode : AAIR, Can_edit : True,	Display parameter soldiers for AAIR. blue sliders to edit	Display parameter soldiers for AAIR. blue sliders to edit	Pass

3.2.5 - Serial Communication Module

list_serial_ports()

Test cases with the expected result, actual result, and pass/fail status for serial ports

id	Test Case	Expected Result	Actual Result	Pass/Fail
1	Plugging the pacemaker into the computer.	COM6 is in the array.	COM6 is in the array.	Pass
2	Unplug the pacemaker from the computer.	COM6 is not in the array.	COM6 is not in the array.	Pass

send_packet()

Test cases with the expected result, actual result, and pass/fail status for sending packets

id	Test Case	Expected Result	Actual Result	Pass/Fail
1	Values = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	Sends same values	Sends same values	Pass
2	Values = [1, 2, 3]	Sends same values	Sends same values	Pass

receive_packet()

Test cases with the expected result, actual result, and pass/fail status for receiving packets

1	Values = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	Returns same values	Returns same values	Pass

	0, 0, 0, 0]			
2	Values = [1, 2, 3]	Does not return values	Does not return values	Pass

get_egram_data()

Test cases with the expected result, actual result, and pass/fail status for obtaining Egram data

id	Test Case	Expected Result	Actual Result	Pass/Fail
1	Values_to_return = [2.4, 9.5]	Returns the same values	Returns the same values	Pass
2	Values_to_return = [2.4]	Does not return values	Does not return values	Pass

4.0 - Assurance Case

Assurance case layed out in a table format

Claims		Context and Assumption	Strategy and Argument	Evidence
1 - Top Claim	The Pacemaker and DCM harmonized device technology is safe for use by patients and physicians	Assumption: ● The pacemaker shall be built using non-toxic material. ● The pacemaker shall be connected to power for functionality. ● The users of the pacemaker and DCM comply with the proper treatment of the physical device itself and do not intend to damage the device. ● The DCM does not encounter any new updates regarding the coding language used to create it, resulting in a crash ● The user is using a device that can allow for a pacemaker connection Context: ● The pacemaker and DCM shall follow the specified requirements (section 1.0). ● A pacemaker and DCM are used in conjunction in a medical setting to help pace a patient's heart. ● Intended to be implanted into the user	Strategy: ● Identify all possible points where breach of security may occur or if a defect is apparent Argument: ● All DCM hazards and defects have been identified and mitigated (1.1) ● All pacemaker hazards and defects have been identified and mitigated (1.2) ● All potential hazards and defects of the pacemaker and DCM harmonized device have been identified and mitigated for communication (1.3)	
1.1 - Top Subclaims	The DCM is safe and secure for intended use	Assumption: ● The pacemaker is used for its	Strategy: ● Test for each possible state of the	

		intended purpose and it is not used inappropriately by both the patient and physician. ● The DCM is used on a device which can support the Python programming language as well as any necessary libraries Context: ● The DCM (device control monitor) is used as a front-end UI to allow for the user to set parameters and select modes to be paced to the pacemaker. ● The DCM hides any unnecessary information from the user and only shows what is necessary to show	DCM through rigorous regression testing Argument: ● All potential unexpected states of the DCM have been identified and mitigated regarding pacemaker connection and permissions (1.1.1) ● The DCM identifies potential areas of security breach for saved user data (1.1.2) ● The DCM utilizes the data received from the pacemaker in a meaningful way and at a pace which does not slow down either the DCM or the pacemaker (1.1.3) ● The DCM only allows for the modiciation of pacemaker parameters by authorized personnel only (1.1.4) ● The DCM only allows for a specific set of inputs for parameters (1.1.5) ● The DCM's requirements are complete, concise, and correct (1.1.6)	
1.1.1 - Subclaim	The DCM has a present behaviour for all expected states, and all other states that do not have a set behaviour, shall be unreachable to the user.	Assumption: ● The clients are unable to modify or alter the file (adequate information hiding is implement) Context: ● Unexpected states of the DCM may include those that happen in a specific sequence of events	Strategy: ● Conduct large scale regression testing to identify all possible states of the DCM ● Any states of the DCM that are too dangerous to deal are to be avoided completely Argument:	● 3.2.2 - Main App Module Test Cases

		• Making a state unreachable means that precautions have been taken to prevent the user from ever reaching a certain state	• The DCM has an expected behaviour for a given state that the user may enter into	
1.1.2	The DCM has a secure user system that avoids data breach, and is difficult to hack into. Any lost information can be retrieved to avoid the loss of an account.	Assumption: • The user selects proper credentials to sign up for an account • The admin password is only known by the admin who is a qualified physician Context: • The user registers using a username, password, email and password confirmation • Maximum of 10 users can be stored	Strategy: • Implement and test a method of encrypting the user's data to avoid being analyzable from third party groups Argument: • The DCM is safe an unbreachable to hackers • The DCM allows for the recovery of lost user data and accounts • The DCM allows for only 1 user of a username to be registered at a time	• 3.2.1 - Global Function Test Cases : Encrypt_password() and decrypt_password() • Forgot password button • User_sign_up_check to check if any existing users share a username
1.1.3	The DCM is safe and monitors and displays relevant information to the user regarding the pacemaker's state, connectivity and data received.	Assumption: • The DCM will display all relevant information so that the user understands the state of the pacemaker without knowing more than they need to Context: • State of the pacemaker relates to its connection, battery levels, parameter, and electrogram data	Strategy: • Identify any important information that the user requires and only display that Argument: • The user does not need to know more than they require to allow for their knowledge to properly maintain and run the pacemaker device • Information and hardware hiding	• Implemtnation of the pacemaker connected, pacemaker battery, parameters verification and electrogram data
1.1.4	The DCM is safe and protects pacemaker functionality from	Assumption: • Regular users do not have access to	Strategy: • Implement a firewall protection to	• Addition of admin password

	unauthorized personnel.	admin permissions without the presence of a professional Context: ● Based on the requirements, only authorized personally shall have access to editing the pacing and parameters on the pacemaker	a lot of the functionality of the pacemaker Argument: ● Regular users shall not have access to editing pacemaker parameters and only verified personnels shall	to lock out button functionality ● 3.2.2 - Main App Module Test Cases include admin permissions
1.1.5	The DCM is secure and mitigates unexpected parameter inputs by the user	Assumption: ● The user inputting/editing parameter data is logged in as Admin ● Is a registered physician or specialized personnel Context: ● Parameter values for the pacemaker must be an element of a specific set of possible inputs.	Strategy: ● Force the user to input a valid parameter using sliders instead of textboxes where parameters must be manually enetered Argument: ● The user managing the DCM and pacemaker can only inpuit preset values and deviations from these preset values is impossible	● Implementation fo sliders that relate to indices of an array to represent the value of a parameter
1.1.6	The DCM satisfies all DCM requirements stated in section 1.0 - Requirement Specification	Assumption: ● Meeting all of the requirements listed that helped to build this device shall leave users and stakeholder content with the finished product Context: ● Requirements are used to help guide the design and iterative process of developing software ● The requirements in 1.0 are a rough set of requirements that were kept in mind when developing this device	Strategy: ● Follow the requirements specified in section 1.0 closely throughout development Argument: ● All requirements are met and implemente throughout the design and development of this device	● Refer to section 2.2 - DCM Design for implementation of requirements into design

1.2 - Top Subclaim	The pacemaker/Simulink is safe and secure for intended use	Assumption: ● The device is used for its intended purpose and there is no malintent when using the device ● The clients are unable to modify or alter the file (adequate information hiding is implement) Context: ● The lookup tables implemented in the serial communication subsystem act as encryption for the system and therefore eradicate the issue of data manipulation ● The simulink model and by extension the pacemaker hardware only communicate with the DCM directly and therefore there is no avenue for any modifications to be done to the pacemakers settings or functions	Strategy: ● Identify all possible points where breach of security may occur or if a defect is apparent Argument: ● The Simulink requirements are complete, concise, and correct (1.2.1) ● All Simulink hazards and defects have been identified and mitigated (1.2.2) ● Simulink and pacemaker only communicate with the DCM (1.2.3)	
1.2.1 - Subclaim	The simulink model/pacemaker follow the outlined requirements in section 1.0 of the documentation	Assumptions: ● Meeting all of the requirements listed that helped to build this device shall leave users and stakeholder content with the finished product Context: ● Requirements are used to help guide the design and iterative process of developing software	Strategy: ● Follow the requirements specified to ensure a complete end product that is consistent with the needs of the product Argument: ● All requirements have been successfully met	● Refer to design decisions section
1.2.2 - Subclaim	The simulink model will not allow the lower rate limit or	Assumption: ● The LRL and MSR values set in the	Strategy: ● Do not let these elements be	● Refer to the design decisions

	maximum sensor rate to be breached under any circumstance	DCM are appropriate for the patient Context: ● Only a healthcare professional can access the DCM to change the parameters for the pacemaker	variable in the pacemaker software Argument: ● These parameters are used as constants and hard limits in conditionals throughout all of the pacemaker pacing modes	of the rate adaptive system ● Implemented strict conditional elements into the stateflow of the rate adaptive pacing modes that do not exceed the LRL and MSR thresholds ● For non-pacing modes, a hard limit is set in the stateflows to again mitigate the risk of this occuring
1.2.3 - Subclaim	Simulink and pacemaker only communicate with the DCM for any changes	Assumption: ● The pacemaker hardware is functioning correctly and there is no faulty equipment Context: ● Only a healthcare professional should be able to access the DCM to change the parameters for the pacemaker	Strategy: ● Enlist some sort of encryption into the serial communicattion Argument: ● Indexes of arrays are sent through the serial ports to ensure no data leakage occurs and without the array data stored on the pacemaker itself the index values tell you nothing	● Lookup tables add a layer of security to support data privacy as outlined in design specifications
1.3 - Top Claim	The pacemaker and DCM can communicate safely between one another.	Assumption: ● The device is used for its intended purpose and there is no malintent when using the device	Strategy: ● Identify all possible points where breach of security may occur or if a defect is apparent	● The pacemaker and DCM accurately and safely receives and sends data in

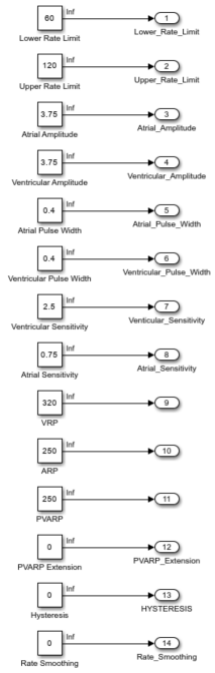
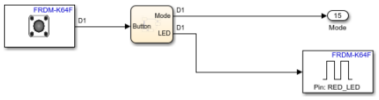
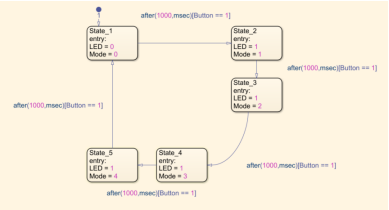
		Context: ● Once the pacemaker is connected, the packets can be sent safely and securely between the pacemaker and DCM	Argument: ● The DCM securely reads and receive parameter data from the pacemaker through serial communication. (1.3.1) ● The DCM and pacemaker can accurately interpret when to be in a read or write state. (1.3.2) ● Data conversion and interpretation is consistent across both the DCM and pacemaker (1.3.3) ● The DCM can safely display the current state on the connection of the pacemaker and any other outstanding states which may compromise the utility (1.3.4)	the Simulink file.
1.3.1 - Subclaim	The DCM and pacemaker has a safe encryption system that allows for secure reading and receiving.	Assumption: ● There is no malicious change of values in the Simulink file. Context: ● Once the DCM or pacemaker sends values, it is encrypted so any external communications in between does not know the context of the packet.	Strategy: ● Identify read and write data information and positioning between ● Examine packet information prior to translation Argument: ● The packet is successfully translated and has no context without the array dictionaries	● 3.2.5 - Array positioning data system used when communicating packets.
1.3.2 - Subclaim	The DCM and pacemaker has a safe and reliable identification system for reading and writing data.	Assumption: ● The DCM and pacemaker communication sync bytes remain consistent, and not changed due to external users.	Strategy: ● Identify differences in packet information sent when reading and writing data Argument:	● 3.2.5 - Sync bytes allow for either pacemaker or DCM to understand when to read or write

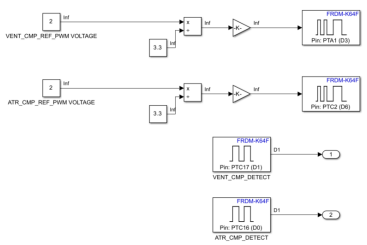
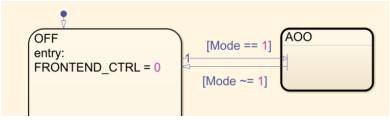
		Context: ● Once values are produced in the serial port, either the DCM or pacemaker can choose to read or write the data correctly.	● The DCM and pacemaker successfully differentiates read and write modes	data.
1.3.3 - Subclaim	The DCM and pacemaker have a safe and secure dictionary for packet translation.	Assumption: ● The dictionary arrays for either the pacemaker or DCM are not changed due to malicious hacking. Context: ● Once a packet is received, the values are accurately translated to what they were supposed to be.	Strategy: ● Identify translations of bytes in both the pacemaker and the DCM Argument: ● The translations of the same values in the DCM and pacemaker are the same.	● 3.2.5 - Each array library for positioning of various parameters is consistent between the DCM and Simulink.
1.3.4 - Subclaim	The DCM has a safe system to confirm the connection between the pacemaker.	Assumption: ● The communication port is physically working properly. Context: ● Once the pacemaker connects or disconnects, the DCM sends a prompt identifying so	Strategy: ● Identify states of connection between the DCM and pacemaker Argument: ● The DCM safely displays connection when the pacemaker is connected, and displays disconnection when the pacemaker is disconnected	● 3.2.2 and 3.2.5 - Pacemaker connected and communication port id is correctly identified

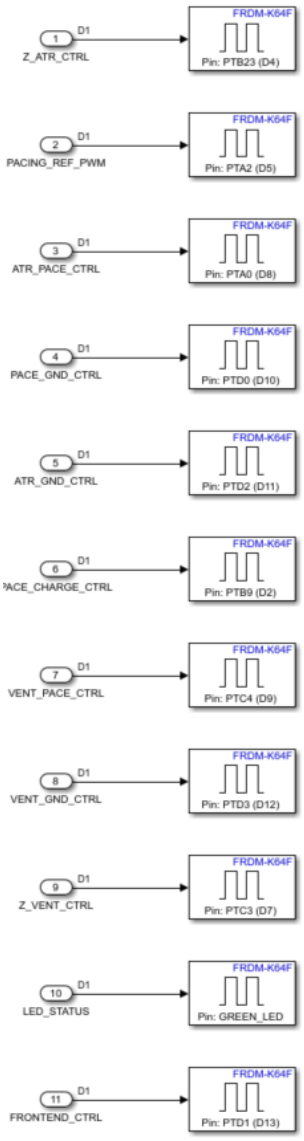
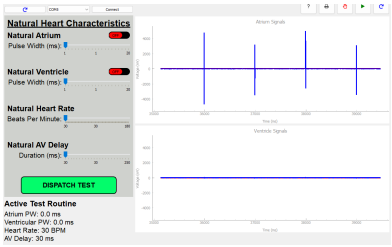
5.0 - Development History

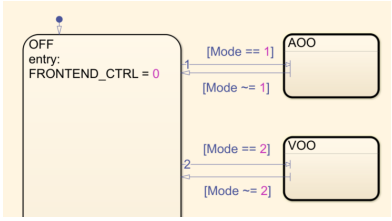
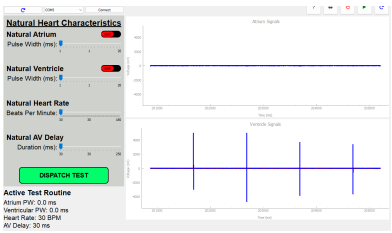
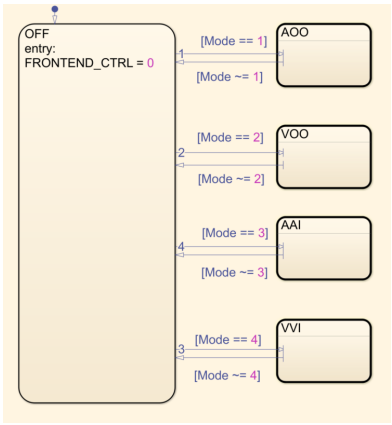
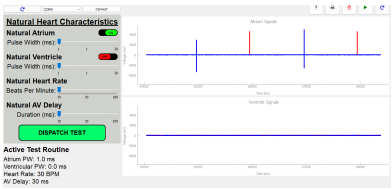
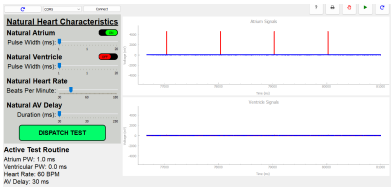
5.1 - Pacemaker Development

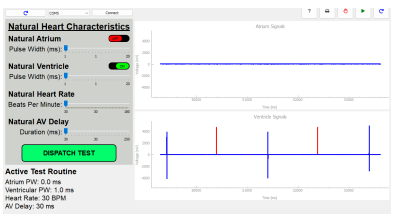
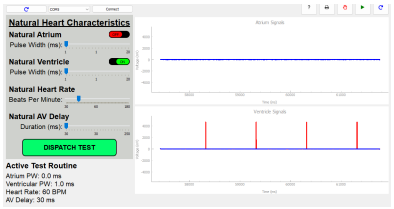
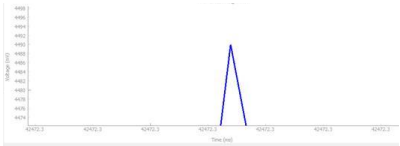
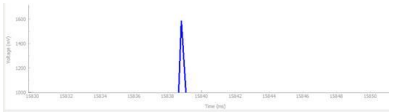
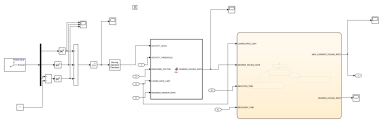
Pacemaker development history.

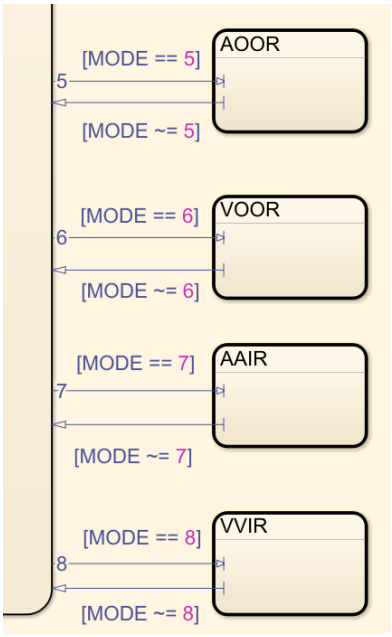
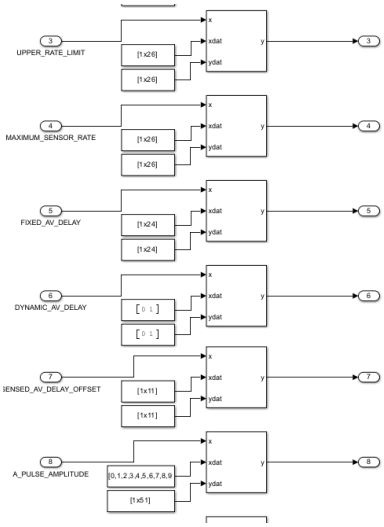
Iteration	Changes	Visual Aid
Friday, October 6, 2023	Started implementing the programmable parameter subsystem. Added a mode selecting chart that takes SW3 as input to switch between modes “OFF”, “AOO”, “VOO”, “AAI” and “VVI”. **Verification:** To verify the operation of the mode selection, a red LED output was added. System starts with the LED off, LED turned on stays on for the next 4 push button presses, LED turns off on the 5th press to complete the cycle.	 *Programmable parameter subsystem.*  *Mode selection system.*  *Mode selection state chart*

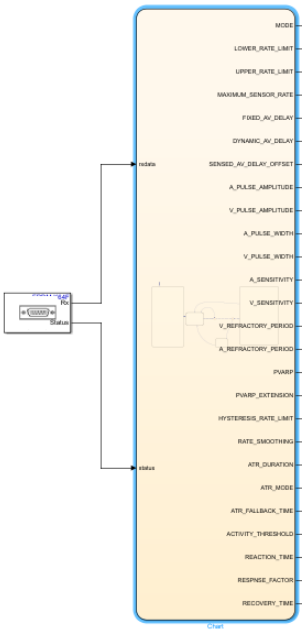
Saturday, October 7, 2023	Implemented the sensing subsystem . **Verification:** FRONTEND_CTRL Pin: PTD1 (D13) was set to 1 to turn on the sensing circuit on the pacemaker. An LED was attached to the Ventricle (D1) and Atrium (D0) individually to detect a pulse from Heartview.	 *Sensing subsystem.*
Sunday, October 8, 2023	1. Implemented mode state flow chart to take input from programmable parameter subsystem and sensing subsystem . Completed “AOO” mode. 2. Implemented Microcontroller input subsystem which took outputs from mode stateflow chart as inputs sent to the microcontroller. **Verification:** Testing unsuccessful. Paces not seen on Heartview.	 *AOO mode.*

		 *Microcontroller input subsystem.*
Wednesday, October 11, 2023	1. PACING_REF_PWM output to microcontroller changed to PWM output instead of digital output. Atrium pin fixed with Matlab code provided in tutorial 1.2. **Verification:** "AOO" mode testing successful. Atrium paces seen on Heartview .	/src/mw_sdk_interface.c'); ii. Within the file that is opened, find the following line: { GPIO_MAKE_PIN(GPIOA_IDX, 0), MW_NOT_USED},// PTA0, D8 and replace it with { GPIO_MAKE_PIN(GPIOC_IDX, 12), MW_NOT_USED},// PTC12, D8 *Do not get any of the letters wrong.* Double click the entry from the *Tutorial 1.2 Matlab troubleshoot code.*  *AOO verification.*

	2. Implemented “VOO” mode . **Verification**: “VOO” mode testing successful. Ventricle paces seen on Heartview.	  *VOO verification.*
Thursday, October 12, 2023	1. Implemented “AAI” and “VVI” modes.	 *Addition of AAI and VVI modes to mode stateflow chart.*
Friday, October 13, 2023	**Verification:** **Test 1:** “AAI” mode was tested with heart rate set to 30BPM and pacemaker lower rate limit set to 60BPM. Test successful. Pacemaker triggered pulses between each normal pulse . **Test 2:** “AAI” mode was tested with heart rate set to 60BPM and pacemaker lower rate limit set to 60BPM. Test successful. Pacemaker did not trigger a pulse. **Test 3:** “VVI” mode was tested with heart rate set to 30BPM and	 *Heart rate 30 BPM.*  *Heart rate 60 BPM.*

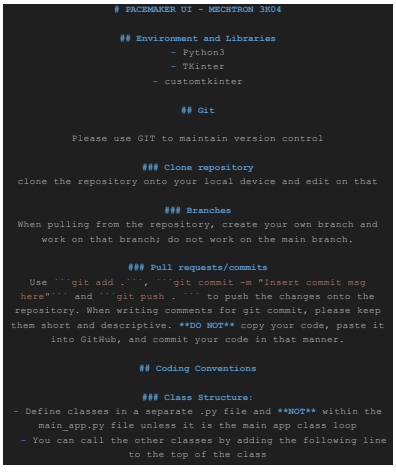
	pacemaker lower rate limit set to 60BPM. Test successful. Pacemaker triggered pulses between each normal pulse . **Test 4:** "VVI" mode was tested with heart rate set to 60BPM and pacemaker lower rate limit set to 60BPM. Test successful. Pacemaker did not trigger a pulse.	 *Heart rate 30 BPM.*  *Heart rate 60 BPM.*
Sunday. October 15, 2023	**Verification:** **Test 5:** Ventricle amplitude was set to 3.75V and 1V to confirm operation. Test successful. Reducing the amplitude voltage resulted in resulted in a lower peak pace amplitude.	 *Ventricle programmable amplitude 3.75 V.*  *Ventricle programmable amplitude 1.00 V.*
Monday, November 13, 2023	**Rate Response Function** Completed the rate response function. Added functionality to decompose accelerometer data and analyze it depending on the activity coming from the board.	 *Rate response function*

Friday, November 17, 2023	**Rate Adaptive Modes** Completed the stateflows for each of the 4 rate adaptive modes, allowing for adpative pacing in the required modes.	 *The 4 rate adaptive modes*
Wednesday, November 22, 2023	**Lookup Tables for Serial Communication** Implemented the indexing of an array feature using the lookup tables for all of the programmable parameters	 *Some of the lookup tables*

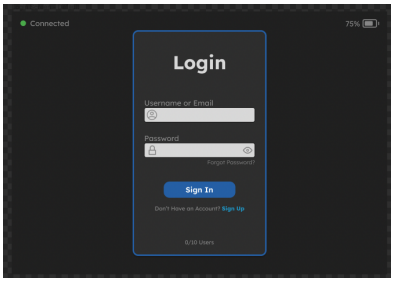
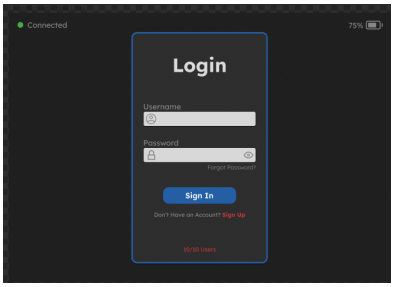
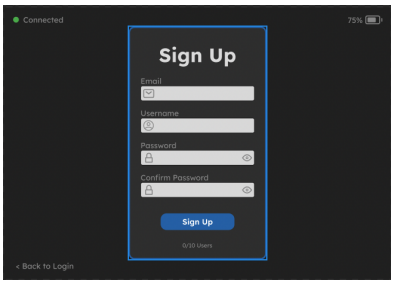
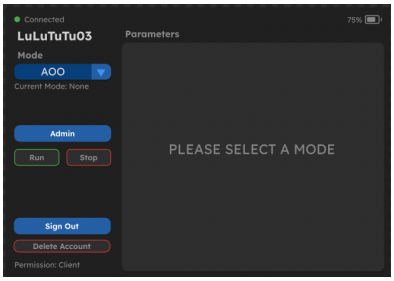
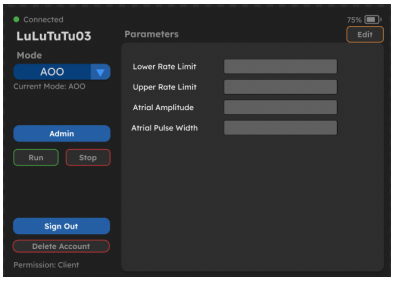
Monday, November 27, 2023	**Completed the Serial Communication Stateflow** Finished implementing the serial communication link between simulink lookup tables and data recieved from DCM	 *Serial communication stateflow chart completed*

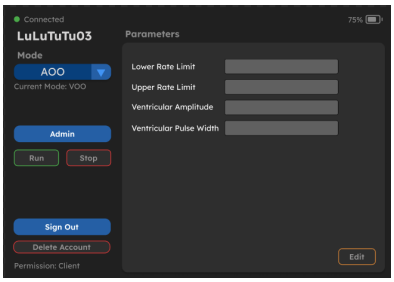
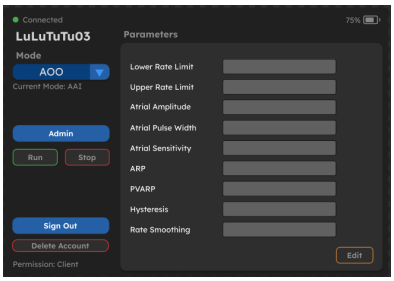
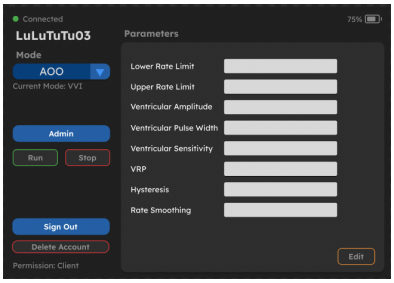
5.2 - DCM Development

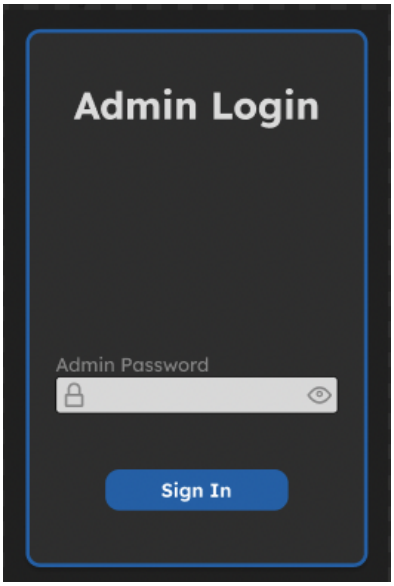
DCM development history

Iteration	Changes	Visual Aid
Thu Oct 5 12:59:07 2023	Created a README.md file for how to organize and name variables and new files.	README.md > # PACEMAKER_MECHTRON_3K04 1 # PACEMAKER_MECHTRON_3K04 2 Repository for pacemaker 3 *Creation of the README file. (Thu Oct 5 12:59:07 2023)*
Thu Oct 5 14:00:09 2023	Finished README.d file, highlighting organization and logistics of code.	

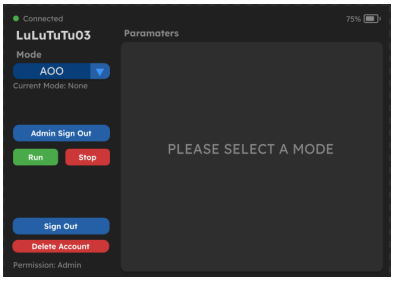
		``py from <file_name> import <class_name> ``` **### Variable Naming:** - For Tkinter variables, name them appropriately based on the type of object it is (ie. button, frames, etc), where in the app it is supposed to be located (ie. welcome screen, main interface), and what the button is meant to represent - Example: Sign In button on welcome screen --> btn_welcome_signin ****Naming Convention Table:**** Tkinter Object | Variable Prefix --- | --- Button | btn Frame | frm Window | win Scroll Bar | sb Label | lbl - For other generic variables that are not objects, include the datatype of the variable within the variable name and what it is meant to do - Example: integer counter --> Variable Name: int_counter - Example: string username --> Variable Name: str_username ****Naming Convention Table:**** Type | Variable Prefix --- | --- Integer | int String | str Boolean | bool Float | flt List | lst **### Using CustomTkinter:** - When creating objects in customtkinter and passing in arguments for the object (Ex. root, text, etc), please include the variable of the argument. Ex. ```py btn_welcome_signin = customtkinter.CTkButton(master=root, width = 191, height=43, text="Sign In", command=response, font=font1) ``` ****NOT**** ```py btn_welcome_signin = customtkinter.CTkButton(mroot, 191, 43, "Sign In", response, font1) ``` **### Comments:** PLEASE COMMENT EVERYTHING FOR CODE YOU WRITE - it will be easier to follow along with what you did and how you did it *Updated README File with organization and logistics of code. (Thu Oct 5 14:00:09 2023)*
Thu Oct 5 14:20:10 2023	Updating README.md file, adding formalities for functions and private variables.	- For functions, name them with the return type of the function, where the function is meant to be called, and general name: - Example: Function returns int after pressing button 1 --> Variable Name: ```py def int_btn1_response(): ``` - For Private variables within classes, add an "_" prefix to the variable name to change it to a private variable - Example: Private variable x which contains an int value of 5 --> Variable Name: ```py self._int_x = 5 ``` *Updating README.md file with formalities for functions and private variables. (Thu Oct 5 14:20:10 2023)*

Fri Oct 6 19:38:29 2023	Completed the UI design using Figma for a preliminary idea of the login screen, register screen, and the main interface for both the client and admin view. The theme of the UI had many possible colour palettes, but ultimately, a dark theme with blue accents emerged victorious. Then initial design of the programmable parameters included text boxes which were later removed in favour of the sliders as they are easier to use and only work on a specified range.	 *Login screen – default.*  *Login screen - max user.*  *Login screen - register.*  *Main interface - client view.* 

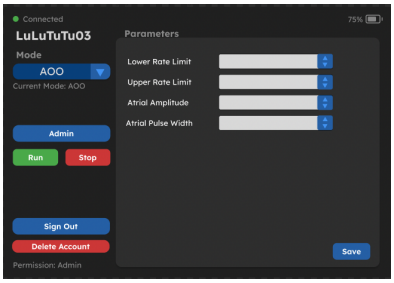
		Main interface: client view – mode: AOO.  *Main interface: client view – mode: VOO.*  *Main interface: client view – mode: AAI.*  *Main interface: client view – mode: VVI.*



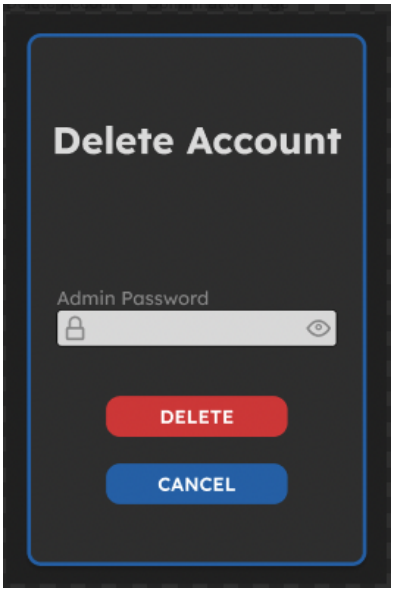
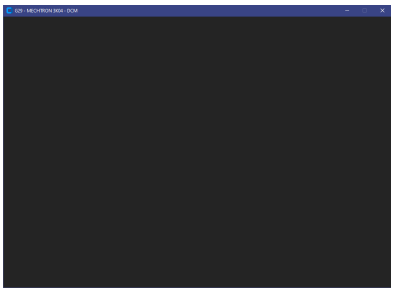
Admin login.

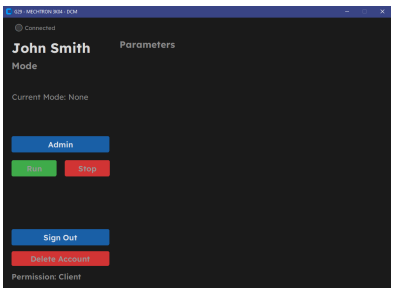
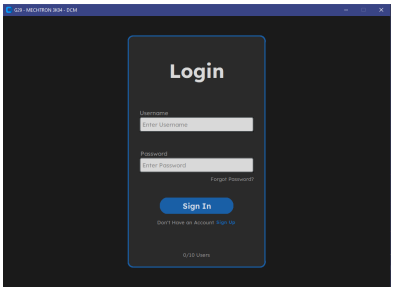
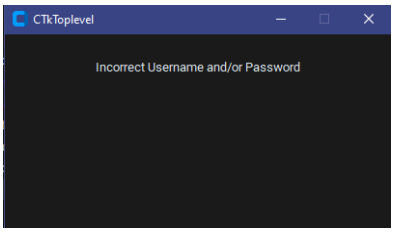
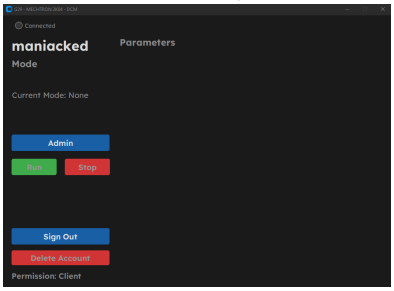


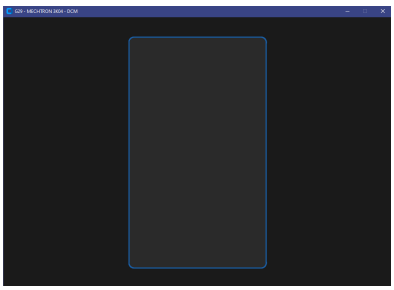
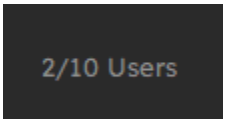
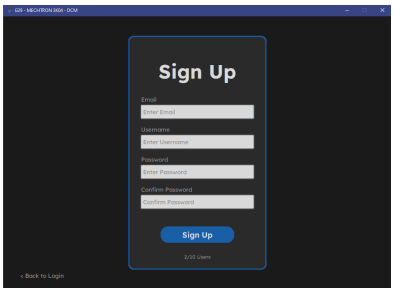
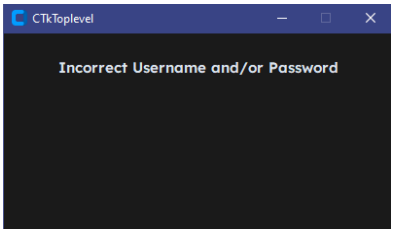
Main interface: admin view.

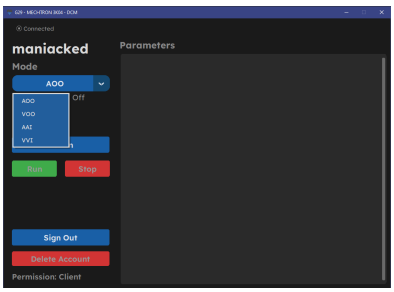
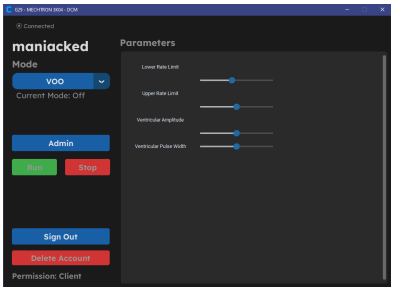
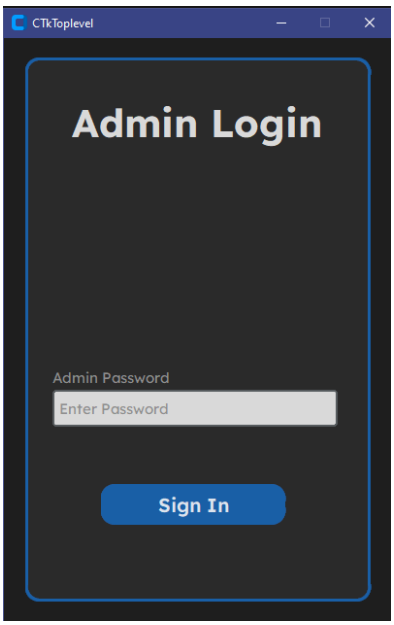


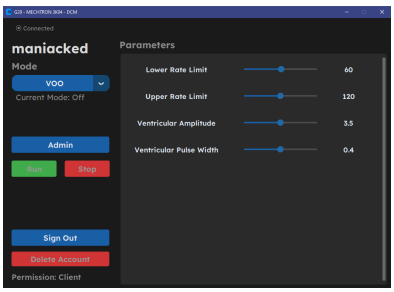
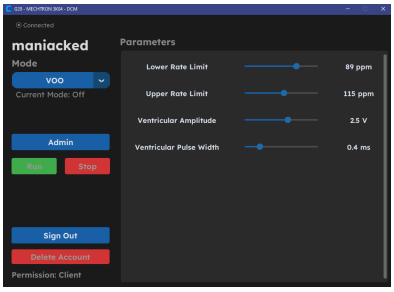
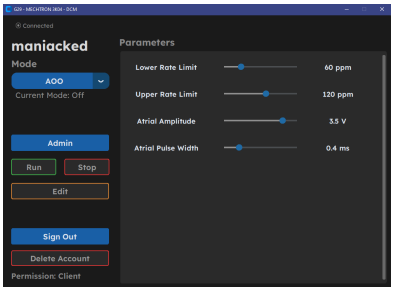
Main interface: admin view – changing parameters.

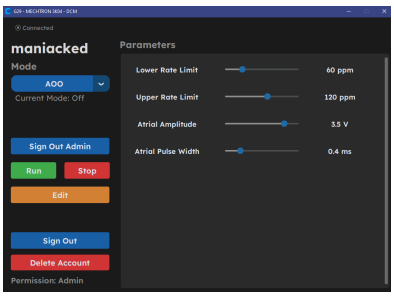
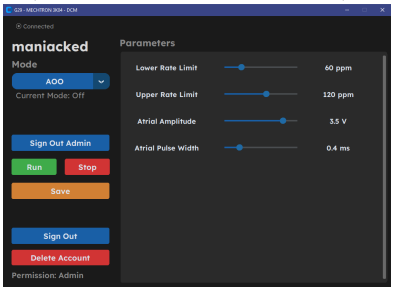
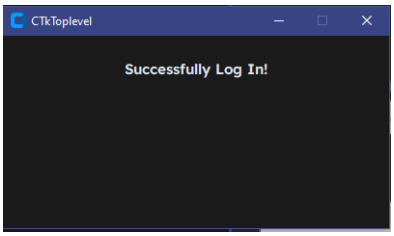
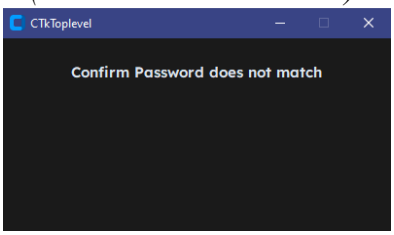
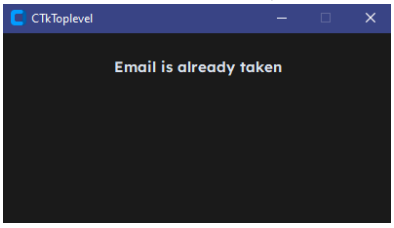
		 *Delete account.*
Sat Oct 7 13:20:23 2023	Created the main user class that will deal with managing user objects. This class was given the functionality to initialize newly registered users using the nominal parameter values for the different modes. The user was also developed to be saved as a dictionary in a JSON file was which was the easiest method to save class objects as there are methods to load json files as dictionaries.	*N/A*
Sat Oct 7 19:11:33 2023	Added main screen box and initialized fonts and colour schemes.	 *Main screen box blank. (Sat Oct 7 19:11:33 2023)*

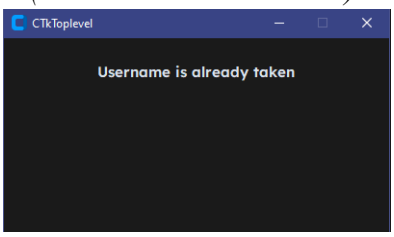
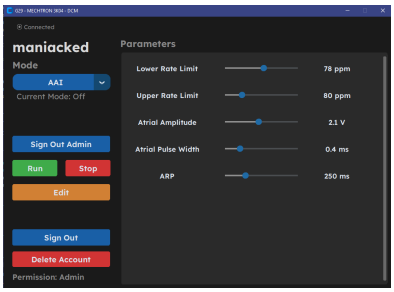
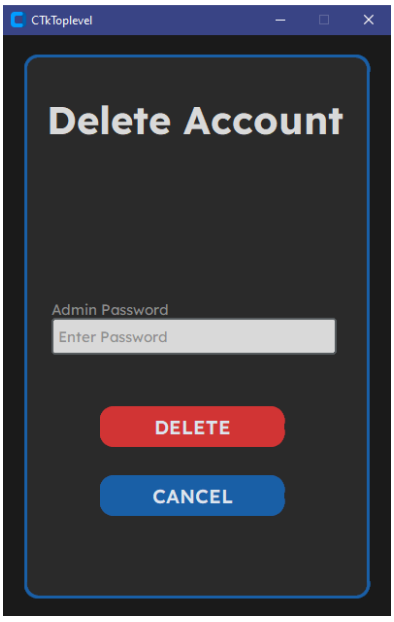
Sat Oct 7 21:06:13 2023	Added buttons and prompts to main interface, including pacemaker connection, name, mode, admin button, run, stop, sign out, delete account and permission. Each button and prompt does not have functionality.	 *Base main interface. (Sat Oct 7 21:06:13 2023)*
Sat Oct 7 21:46:36	Added buttons and prompts to login screen, including username and password entry textboxes, forget password button, sign up button, sign in button and visual indicator of active user count. None of these features have functionality.	 *Base login screen. (Sat Oct 7 21:46:36)*
Sat Oct 7 23:40:01	Added functionality to login screen, including pop up window when username or password is incorrect, and functional sign in button when correct credentials are inputted. Added functionality to main interface, including updating the name and permissions of the logged in user.	 *Incorrect username and/or password popup. (Sat Oct 7 23:40:01)*  *Updated main interface with functionality. (Sat Oct 7 23:40:01)*

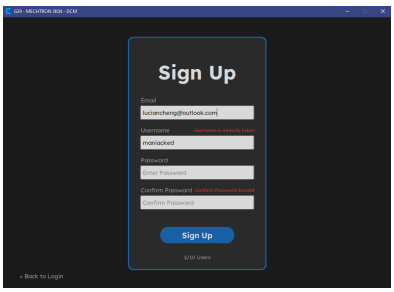
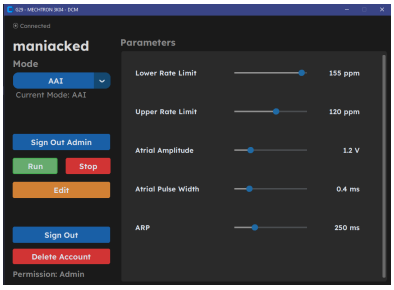
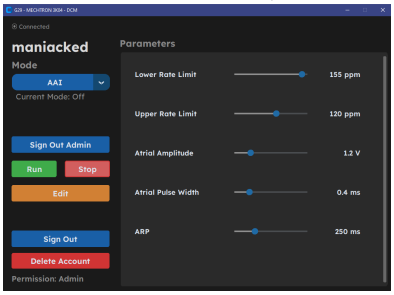
Sun Oct 8 12:11:54 2023	Added functionality to main interface, including letting a user sign out and redirect them to the log in screen using sign out button. Added functionality to login screen, including redirection to the registration page using the sign up button and live updated count on active users. Created template for registration page.	 *Blank registration screen. (Sun Oct 8 12:11:54 2023)*  *Active user count. (Sun Oct 8 12:11:54 2023)*
Sun Oct 8 12:25:07	Added new icon cause it's cool.	 *Logo change. (Sun Oct 8 12:25:07)*
Sun Oct 8 15:09:19 2023	Added buttons and prompts to registration screen, including textbox entries for email, username, password, and confirm password. Sign up button functionally creates a new json file if given credentials that does not exist in the system. User notification of existing credentials are not accurate. Back to login button is functional, navigating a user back to the login screen when selected.	 *Base registration screen. (Sun Oct 8 15:09:19 2023)*  *Visually updated incorrect username and/or password prompt. (Sun Oct 8 15:09:19 2023)*

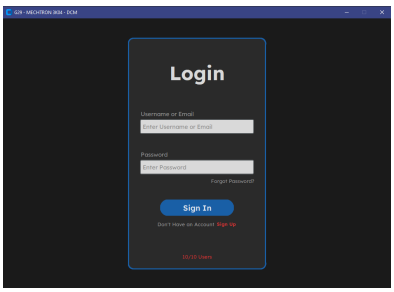
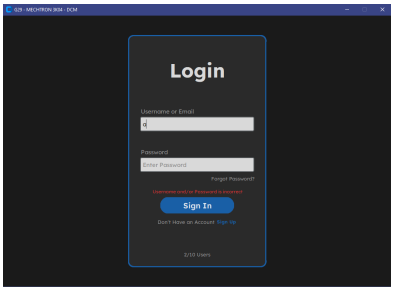
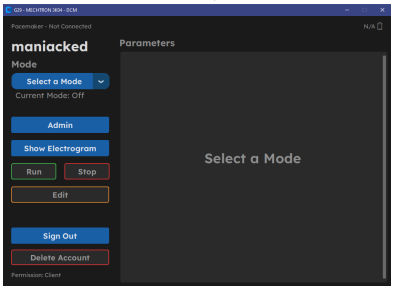
Sun Oct 8 15:25:31 2023	Added functionality to main interface, including ability to navigate through different pacemaker modes through the dropdown menu. Different arrays of values are viewed for each different mode in the terminal.	 *Updated main interface with dropdown mode selection. (Sun Oct 8 15:25:31 2023)*
Sun Oct 8 17:19:21 2023	Added functionality to main interface, including scroll selection for specific parameters for each specific mode with correlating parameter titles.	 *Updated main interface with scroll selection. (Sun Oct 8 17:19:21 2023)*
Sun Oct 8 17:19:40 2023	Added functionality to main interface, including letting the user navigate to admin login page using admin button. Created an admin login page, including textbox entry for admin password and sign in button. Sign in button has no functionality.	 *Base admin login screen (Sun Oct 8 17:19:40 2023)*

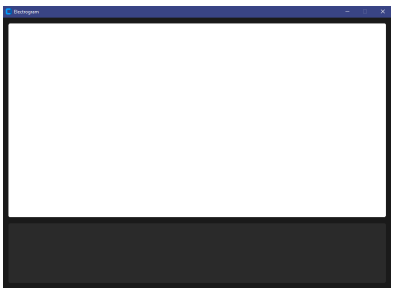
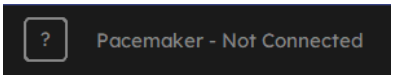
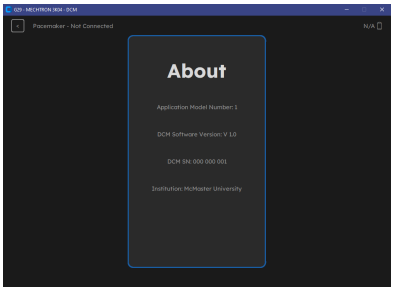
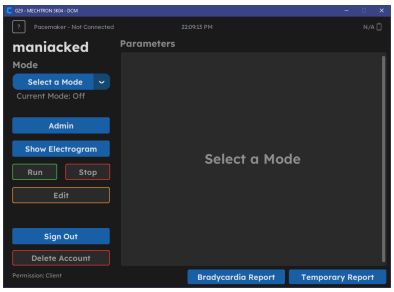
Sun Oct 8 18:49:33 2023	Added functionality to main interface, including uploading nominal values to each mode. Visually updated scroll selection.	 *Updated main interface with nominal values (Sun Oct 8 18:49:33 2023)*
Sun Oct 8 19:16:32 2023	Added functionality to main interface, including adding a maximum and minimum scroll value for each parameter, and correct incrementations of values. Fixed a bug where the incorrect values would be sent from the scrollabale parameter frame module to the main application module due to the initialization of the class. This meant that the data getting transferred was the default values from the user class and not the most updated values from the sliders.	 *Updated main interface with specific values on parameters for each mode. (Sun Oct 8 19:16:32 2023)*
Mon Oct 9 00:20:55 2023	Added functionality to the admin login screen, including the ability to log in as an admin with the correct credentials in the admin login page. Added functionality to the main interface, including adding an edit, sign out admin, and a delete account button. Permissions to click run, stop, edit and delete account are revoked for a client without an admin password. Admins can select edit button to edit each parameters for modes, but save button does not save parameters properly. Admins are able to sign out of admin perminssions using the sign out admin button. Delete account,	 *Updated main interface for client permissions only. (Mon Oct 9 00:20:55 2023)*

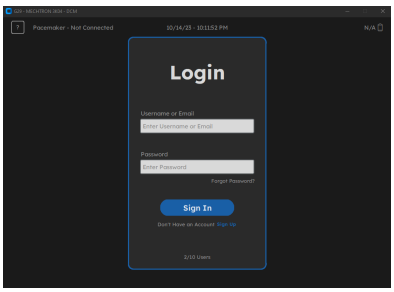
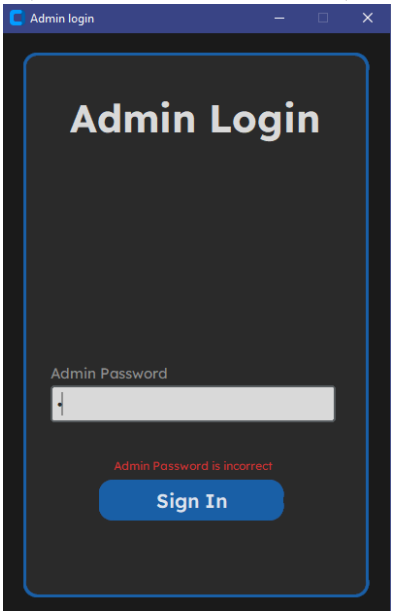
	run, and stop buttons have no functionality.	 *Updated main interface for admin edit permissions only. (Mon Oct 9 00:20:55 2023)*  *Updated main interface for admin save permissions only. (Mon Oct 9 00:20:55 2023)*
Mon Oct 9 01:43:21 2023	Added functionality to registration screen, including the ability to register and create a new json account for a user if credentials requirements. This includes a username and email not taken by another existing account, and password and confirm password that matches and is not blank. Popup windows for successful or unsuccessful registrations were created, and reasons for unsuccessful registrations. Successful window does not display accurate prompt.	 *Successful Registration prompt. (Mon Oct 9 01:43:21 2023)*  *Confirm password does not match prompt. (Mon Oct 9 01:43:21 2023)* 

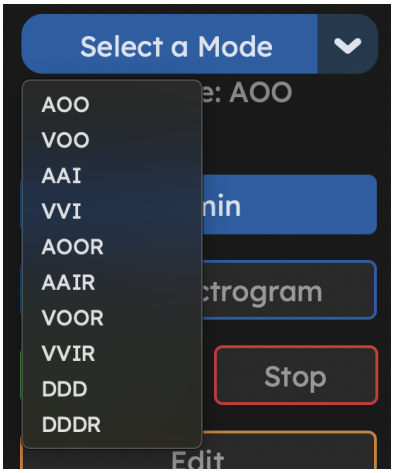
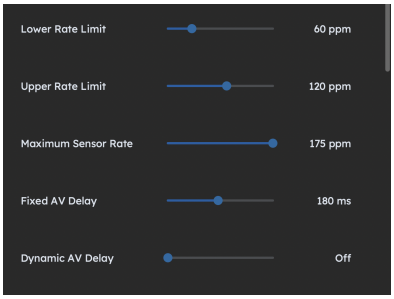
		Email is already taken prompt. (Mon Oct 9 01:43:21 2023)  *Username is already taken. (Mon Oct 9 01:43:21 2023)*
Mon Oct 9 12:30:40 2023	Fixed functionality of main interface, including allowing an admin to save their updated parameters correctly. Resolved a bug where if the edit button was pressed before a mode was selected, the program would inverse the edit and save functionality. This was resolved by creating a variable that lets the scrollable frame module know if the the current mode of the button is set to edit or save.	 *Updated main interface with saving parameter functionality. (Mon Oct 9 12:30:40 2023)*
Mon Oct 9 12:42:09 2023	Added functionality to main interface, including allowing an admin to navigate to a delete account page using the delete account button. Created a delete account page, with prompts and button, including a textbox entry for admin password, delete button, and cancel button. None of these have functionality.	 *Base delete account page (Mon Oct 9 12:42:09 2023)*

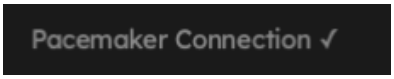
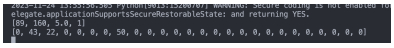
Mon Oct 9 15:25:27 2023	Added functionality to delete user page, including the ability for an admin to delete an account when inputting the correct admin password. Visually updated registration screen by changing pop up prompts for invalid or existing sign up credentials to red text.	 *Updated invalid sign up credential prompts. (Mon Oct 9 15:25:27 2023)*
Tue Oct 10 13:21:30 2023	Added functionality to main interface, including the ability for an admin to run a selected mode when pressing the run button, visually indiciated by the current mode prompt that is updated. As well, an admin can stop the current running mode by pressing the stop button, visually indiciated by the current mode prompt that is updated. Resolved a minor changeg in the justification of text of the scrollable parameters frame. Fixed a bug where if the mode was selected many many times from the drop-down menu, it would lag the application. This happened because the widgets were not correctly getting deleted but only being unplaced. This meant that the application still ahd memory of the widgets. This issue was resolved by destroying the widgets whenever the screen is changed or if a new mode is selected.	 *Updated main interface with run functionality. (Tue Oct 10 13:21:30 2023)*  *Updated main interface with stop functionality. (Tue Oct 10 13:21:30 2023)*

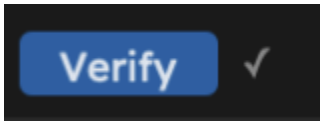
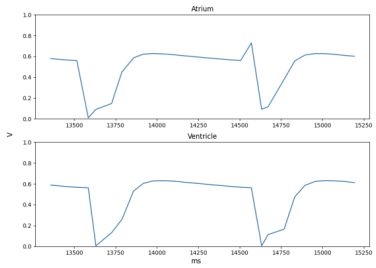
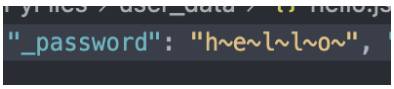
Tue Oct 10 20:15:39 2023	Added functionality to the login screen, including the inability to navigate to the sign-up page when the active user count is at a maximum of 10. The user count and sign up button are visually updated with a red text.	 *Updated login screen with maximum users. (Tue Oct 10 20:15:39 2023)*
Tue Oct 10 20:30:00 2023	Visually updated incorrect credential prompt in the log in screen.	 *Updated incorrect credential prompt for login screen. (Tue Oct 10 20:30:00 2023)*
Thu Oct 12 00:18:56 2023	Added functionality to main interface, including a show electrogram button that navigates any permission to an electrogram page with no functionality. Updated pacemaker connect prompt, and added a battery status prompt with no functionality.	 *Updated pacemaker connection prompt (Thu Oct 12 00:18:56 2023)*  *Updated main interface with show electrogram and battery prompt. (Thu Oct 12 00:18:56 2023)*

		 *Base electrogram screen (Thu Oct 12 00:18:56 2023)*
Thu Oct 12 13:25:15 2023	Added functionality to top bar, including a ? button that users can press to navigate to an about screen. Created an about screen highlighting application model number, DCM Software Version, DCM SN, and institution. Users can navigate back to the previous screen using the < button.	 *Updated about button in login screen (Thu Oct 12 13:25:15 2023)*  *Base about screen (Thu Oct 12 13:25:15 2023)*
Thu Oct 12 14:20:50 2023	Added functionality to top tab, including a clock. Added buttons to main interface, including bradycardia report and temporary report. These buttons have no functionality.	 *Updated main interface with bradycardia report and temporary report buttons. (Thu Oct 12 14:20:50 2023)*

Thu Oct 12 18:46:55 2023	Added functionality to top tab, including date beside the clock. Added functionality to admin login page, including prompt for incorrect credential input for admin password.	 *Updated login screen with date. (Thu Oct 12 18:46:55 2023)*  *Updated admin login screen with an invalid password prompt. (Thu Oct 12 18:46:55 2023)*
Wed Nov 15 15:30:15	Ran into error with saving parameters. The error was due to a type conflict since we kept type as int but we originally used np.arange which uses floats so we changed it to range and did not use the library function.	 *Error experienced with saving parameters (Wed Nov 15 15:30:15)*
Wed Nov 15 16:37:38	Experienced issue where there was one too many slider increments. The solution was to "-1" from the number of steps for the slider initiation.	 *Before the solution was implemented where the slider is at "Off"* 

		Before the solution was implemented where the slider is supposed to be at "On"  *Slider after implementing the solution (Wed Nov 15 16:37:38)*
Thu Nov 16 16:32:11	Added new modes that were required for assignment 2, including AOOR, AAIR, VOOR, VVIR, DDD, DDDR.	 *New modes added (Thu Nov 16 16:32:11)*
Wed Nov 22 15:19:32	Issue with some of the parameters on the GUI are not zero, but in the terminal, they are displayed as zero, which is an issue	 *Outputs in the terminal before solution was implemented*  *Outputs on the GUI before solution was implemented*  *Outputs in the terminal after solution was implemented*

		 *Outputs on the GUI after solution was implemented (Wed Nov 22 15:19:32)*
Wed Nov 22 19:32:22	Added Pacemaker connection status at top left corner of GUI to display the status of the connection between DCM and pacemaker.	 *Status of pacemaker when it is not connected*  *Status of pacemaker when it is connected (Wed Nov 22 19:32:22)*
Fri Nov 24 13:31:28	Ran into issue where failed to initialize mode upon logging in w/o editing parameters. There was a type error since it was initialized to “None”.	 *Error in the terminal before the solution was implemented*  *Output in the terminal after the solution was implemented (Fri Nov 24 13:31:28)*
Fri Nov 24 14:33:19	Problem where array was being updated when “run” button is pressed and “save” button is not pressed. The array should only be updated when the save button is pressed. The solution was to use listname.copy() function in python.	 *Error in the terminal before the solution was implemented (Fri Nov 24 14:33:19)*
Sat Nov 25 9:50:12	Added serial communication between DCM and pacemaker and parameters.	*N/A*

Sat Nov 25 12:43:39	Added toggle buttons based on if pacemaker is connected or not.	 *Verify button when pacemaker is not connected (inactive button)*  *Verify button when pacemaker is connected (active button) and the parameters match (Sat Nov 25 12:43:39)*
Mon Nov 27 17:13:30	Added the Egram graph.	 *Data plotted on Egram graph (Mon Nov 27 17:13:30)*
Tues Nov 27 8:22:11	Added password encryption to improve security and safety.	 *Encrypted password (Tues Nov 27 8:22:11)*