Pacemaker User Guide and Documentation for Simulink and DCM Components

MECHTRON/SFWRENG 3K04 Fall 2024

L03 Group 23: LeTron James

Instructor: Thomas Chiang

Members:

Vineet Aggarwal (400432692)

Mitchell Von Dehn (400454214)

Marcus De Maria (400455619)

Aidan O'Brien (400455496)

Yajat Sharma (400461856)

Nathan Sinha (400452693)

Table of Contents

1. Project Overview

A pacemaker is a small gadget, commonly used to treat arrhythmias. The heart may beat too quickly, too slowly, or irregularly during an arrhythmia. In order to assist your heart in beating at a regular pace and rhythm, pacemakers emit electrical pulses. In order for your heart to pump blood to your body more effectively, pacemakers can also assist your heart's chambers beat in unison. If you have heart failure, you could require this.

Either a short-term or long-term pacemaker may be required. Usually placed through a neck vein, a temporary pacemaker stays outside of your body. Your chest or belly is fitted with a permanent pacemaker. Permanent pacemakers are the main subject of this discussion. You might have to spend a few hours or even overnight in the hospital to receive a pacemaker. After you return home, your doctor might remotely examine your pacemaker and arrange for routine check-ups.

This assignment is divided into two main sections. Developing real-time software for the hardware platform is the initial step. The second component entails accurately recording the progress made throughout the project. This assignment's primary goal is for developers to be able to analyze the given documentation and extract the precise data required to construct a working pacemaker and DCM.

Stateflows for the AOO, VOO, AAI, and VVI pacemaker modes can be implemented using Simulink for a pacemaker in permanent state. The programmable parameters specified in the requirements document should be used by the stateflow. Particularly noteworthy are the rate features (delays and restrictions), the chamber or chambers being paced, and the pulse parameters (width and controlled amplitude).

Along with the pacemaker itself, the project involves designing a DCM that can design and operate on computers. The DCM for this project is based in python, using tkinter as the GUI framework along with various other imports and libraries. For the current state of the project, communication is not required between the Pacemaker and DCM. For now the main focus is applying and demonstrating the presentation layer, or front end, of the DCM application.

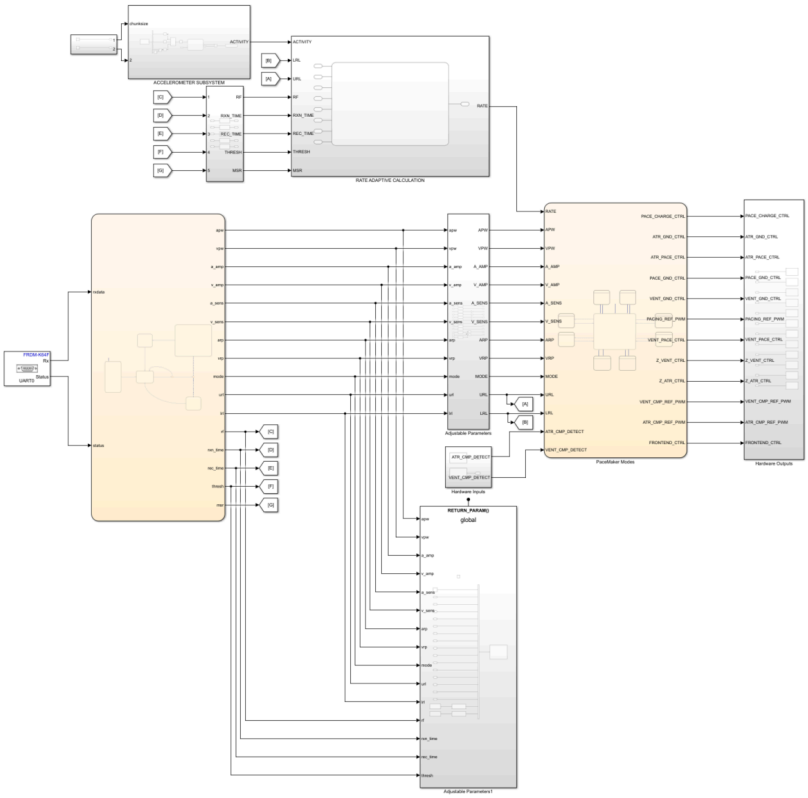
2. Simulink Documentation

2.1 Simulink Overview

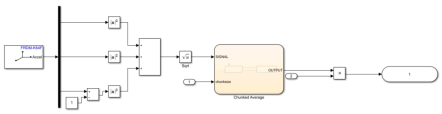
2.1.1 Description

In this project, the end goal is to design, implement, and test a functioning pacemaker software in an embedded environment. As the project is divided into 2 parts, this section of the document will focus on the developments made in part 1 and will describe the different pacing modes in detail. Simulink was used extensively in this part and will also be used in the second part of the project as well.

2.1.2 Images

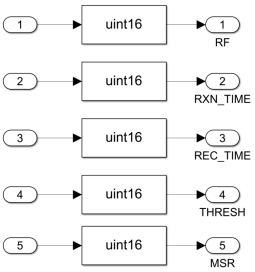


This is a picture of the overall view of the simulink model with the hardware hiding implemented. On the left are the subsystems for the hardware inputs which are the measured variables and the adjustable parameters which are the control variables. There are also the subsystems for measuring the activity in the heart and then converting it into a value that can be used as the changing heart rate. On the right are the hardware outputs, which connect the outputs to the pins on the board. In the middle is the actual stateflow with all of the modes, which will be discussed further in the report.

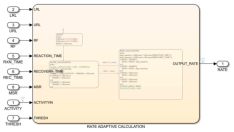


This is an inside view of the accelerometer subsystem where we take the raw input from the accelerometer and filter it into a readable and useable value

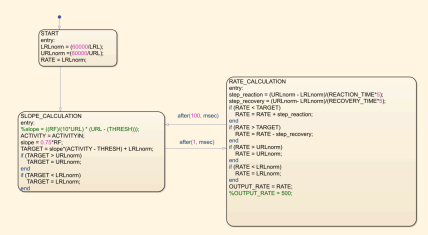
This is the hardware hiding for the chunksize into the accelerometer subsystem to help us calculate a moving average



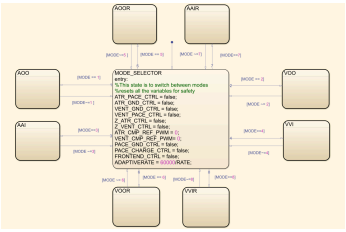
This is an input into the Rate adaptive calculation subsystem, it just shows how we typecast all of the values to make sure that they fit with the calculations that we do inside of the system



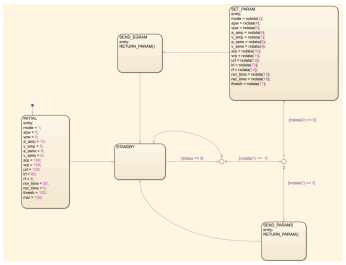
This is the inside of the rate adaptive calculation subsystem, and you can see how we do all of the calculations inside to convert the activity level to an appropriate heart rate value



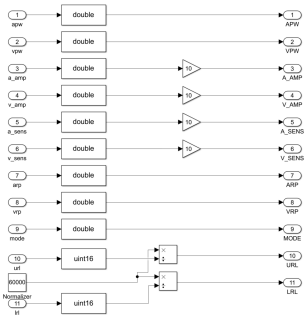
This is an inside view of the subsystem that we show above, contains all of the calculations that we used and the logic behind it as well



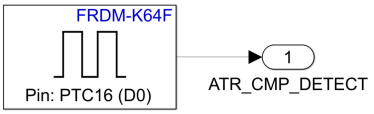
This is the mode selector that we have to switch between pacemaker modes and reset all variables between transitioning states for safety. It also acts as a dead zone if an invalid mode is entered to ensure patient safety. We will show the inside of all of the modes later within this document

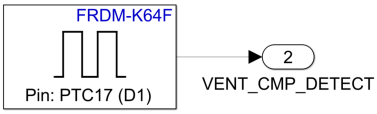


This is an inside of the serial communication subsystem, it shows how and when we update the values inside simulink using what we get from the DCM

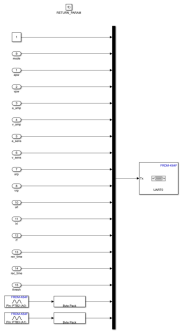


This is the inside view of the adjustable parameters, and shows what values we assigned to each of the variables as well as how we come up with the values, having to normalize the LRL and URL by converting ppm → msec for the code, and making it user-friendly

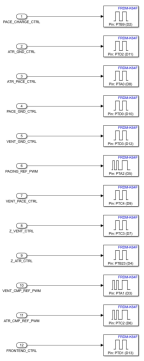




This is the inside of the hardware inputs stateflow showing which pins we measure and what variable we assign the read values to



This is the inside of the adjustable parameters subsystem, where we have the function to send parameters back to the DCM for racing and checking as well as measuring atrial and ventricle signals for the electrogram



This is the inside of the hardware outputs subsystem, and it shows which variables are connected to which pins in relation to the outputs

2.1.3 Safety Features

Since this pacemaker is designed as a life-support device for human installation, safety is of paramount importance. We've implemented a range of measures to ensure that the device operates reliably and securely, mitigating any potential risks to the patient. For example, we have employed a mode selector state, as detailed earlier, which helps manage the various operating modes and provides structured control over the pacemaker's functionality. This mechanism is essential to prevent any unintended behaviors that might arise from improper mode transitions.

Furthermore, we took special precautions to prevent hardware variables from changing in an incorrect sequence, which is critical in avoiding any direct connection of the input PWM (Pulse Width Modulation) signal to the patient. This step helps ensure that the pacing signals are delivered precisely and safely, with no risk of unexpected pulses or errors reaching the patient, which could compromise the device's performance and patient safety.

Looking ahead, several enhancements could further improve the pacemaker's safety and functionality. One such improvement would be the implementation of an external control button. This button could serve as an emergency stop feature, allowing medical personnel or the patient to temporarily disable pacing functions. While the button is pressed, the pacemaker would remain in a "charging" or inactive state, ceasing all pacing functions. This could provide an additional layer of control in critical situations where immediate intervention is required.

Additionally, we plan to incorporate a feature that restricts the input values for constant parameters to within a predefined safe range. This would ensure that any attempt to enter values outside of the acceptable thresholds would be automatically blocked, preventing any accidental or erroneous input that could lead to unsafe operating conditions.

2.2 Variables

2.2.1 Measured Variables

Name	Description/Full Form	Units/Type	Range of Values
t*	**Time** elapsed since the pulse is started	msec	{0,inf}
ATR_CMP_DETECT	Returns whether or not an **atrial pulse is detected**	Boolean	{true,false}
VENT_CMP_DETECT	Returns whether or not an **ventricular pulse is detected**	Boolean	{true,false}

*t is a variable we introduced for the report, it does not show up in the code, it is just present in the report to simplify state diagrams and for ease of understandability and readability

2.2.2 Constant Variables

Name	Description/Full Form	Units/Type	Range of Values / Tolerance
APW	Desired **Atrial Pulse Width**	mSec	{0.1,1.9} / 0.2
VPW	Desired **Ventricular Pulse Width**	mSec	{0.1, 1.9} / 0.2
A_AMP	Desired **Atrial Amplitude** for pulse	mV	{off} {0.5, 3.2} {3.5, 7} / ± 12%
V_AMP	Desired **Ventricular Amplitude** for pulse	mV	{off} {0.5, 3.2} {3.5, 7} / ± 12%
A_SENS	Desired threshold for **Sensing of Atrial** action potential	mV	{0.25, 0.5, 0.75, 1.0-10} ± 20%
V_SENS	Desired threshold for **Sensing of Ventricular** action potential	mV	{0.25, 0.5, 0.75, 1.0-10} ± 20%
ARP	**Atrial Refractory Period** to avoid sensing own pulse	mSec	{150, 500} / ± 8
VRP	**Ventricular Refractory Period** to avoid sensing own pulse	mSec	{150, 500} / ± 8
MODE	To switch modes	Integer	{1,4}
URL*	**Upper Rate Limit** for the patients BPM	Pulses / Minute	{50,175}, >LRL

Name	Description/Full Form	Units/Type	Range of Values / Tolerance
APW	Desired **Atrial Pulse Width**	mSec	{0.1,1.9} / 0.2
VPW	Desired **Ventricular Pulse Width**	mSec	{0.1, 1.9} / 0.2
A_AMP	Desired **Atrial Amplitude** for pulse	mV	{off} {0.5, 3.2} {3.5, 7} / ± 12%
V_AMP	Desired **Ventricular Amplitude** for pulse	mV	{off} {0.5, 3.2} {3.5, 7} / ± 12%
A_SENS	Desired threshold for **Sensing of Atrial** action potential	mV	{0.25, 0.5, 0.75, 1.0-10} ± 20%
V_SENS	Desired threshold for **Sensing of Ventricular** action potential	mV	{0.25, 0.5, 0.75, 1.0-10} ± 20%
ARP	**Atrial Refractory Period** to avoid sensing own pulse	mSec	{150, 500} / ± 8
LRL	**Lower Rate Limit** for the patients BPM	Pulses / Minute	{30,175}, <URL
RATE	The rate at which the pacemaker paces in rate adaptive modes	Pulses / Minute	{LRL, URL}

*We have URL included although for the current modes that we are constructing it was not necessary to implement any usage for the variable and therefore did not

2.3 Modes

2.3.1 AOO

2.3.1.1 Overview

This is the mode solely to pace the atrium at a constant rate, which is the LRL.

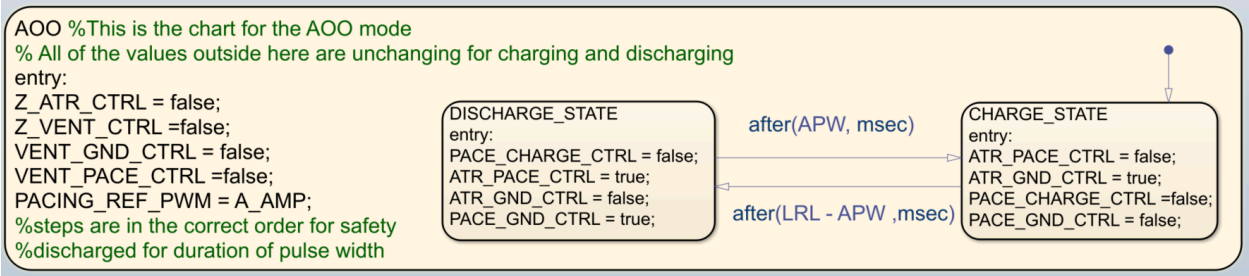
2.3.1.2 Variables

Name	Reference
t	2.2.1
LRL	2.2.2
A_AMP	2.2.2
APW	2.2.2

2.3.1.3 Requirements

Current state	t value	Next state
Charging	t > LRL - APW	Discharging
Discharging	t > APW	Charging

2.3.1.4 Stateflow diagram



2.3.1.5 Testing

Test Case*	Expected Result	Actual Result	Status

Natural Atrium	Natural Ventricle			
OFF	OFF	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass

*The LRL is set to 60P/M

This is the only test case that we need for AOO, because it only has one purpose which is to give a consistent pulse as the set LRL, without sensing any natural pulse.

Heartview outputs for the test case:

Report ID: AOO_Report

Active Test Routine

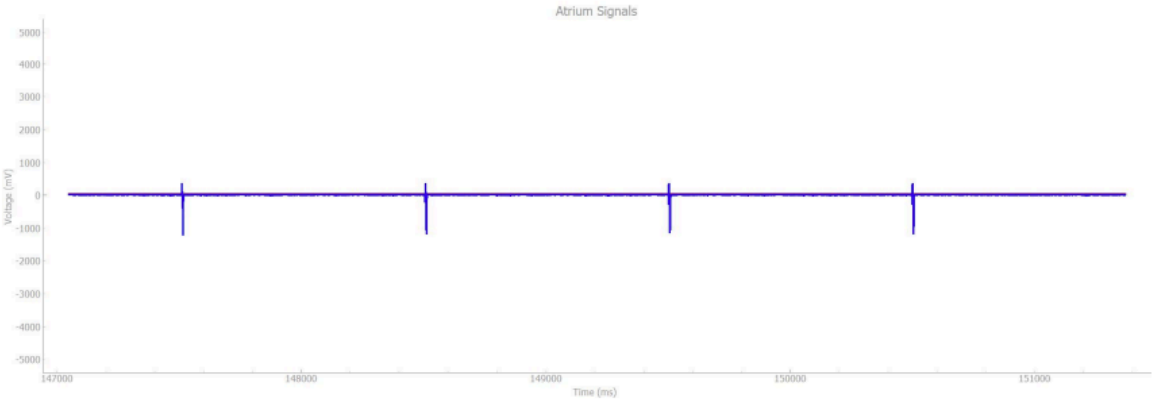
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 60 BPM

AV Delay: 30 ms

Atrium Plot:



2.3.1.6 Future Changes

There are no future changes, the AOO mode is complete.

2.3.2 VOO

2.3.2.1 Overview

This is the mode solely to pace the ventricle at a constant rate, which is the LRL.

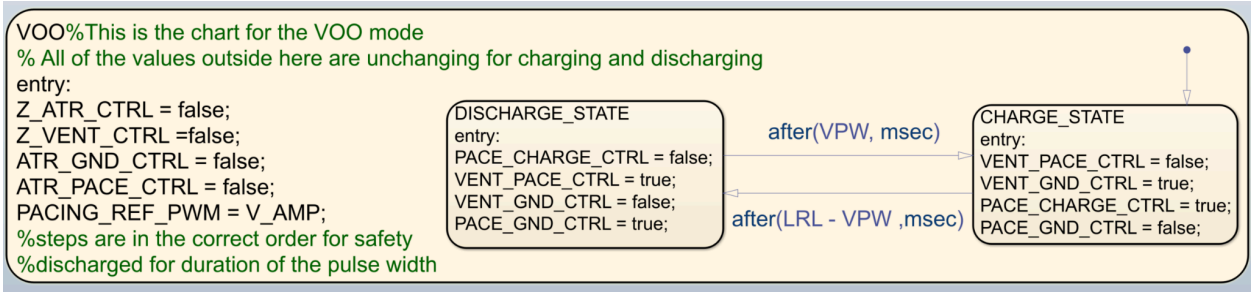
2.3.2.2 Variables

Name	Reference
t	2.2.1
LRL	2.2.2
V_AMP	2.2.2
VPW	2.2.2

2.3.2.3 Requirements

Current state	t value	Next state
Charging	t > LRL - VPW	Discharging
Discharging	t > VPW	Charging

2.3.2.4 Stateflow diagram



2.3.2.5 Testing

Test Case*		Expected Result	Actual Result	Status
Natural Atrium	**Natural Ventricle**			
OFF	OFF	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass

*The LRL is set to 60P/M
This is the only test case that we need for VOO, because it only has one purpose which is to give a consistent pulse as the set LRL, without sensing any natural pulse.

Heartview outputs for the test case:

Report ID: VOO_Report

Active Test Routine

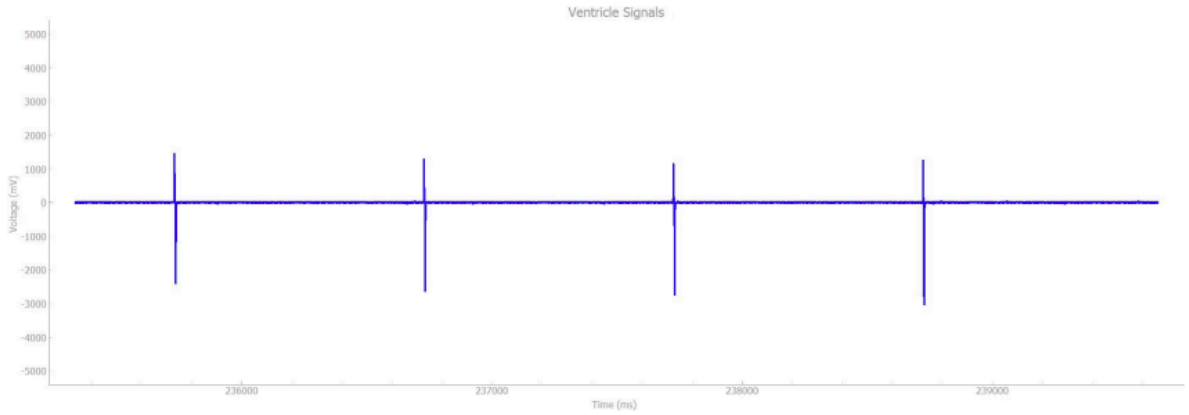
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 60 BPM

AV Delay: 30 ms

Ventricle Plot:



2.3.2.6 Future Changes

There are no future changes, the VOO mode is complete.

2.3.3 AAI

2.3.3.1 Overview

This is the mode to first sense the atrium and in response to the natural atrium pulse, inhibit the pacemaker pulse, or in lack of atrial pulse let the pacemaker pulse the heart. (Atrial Pacing, Atrial Sensing, Inhibiting)

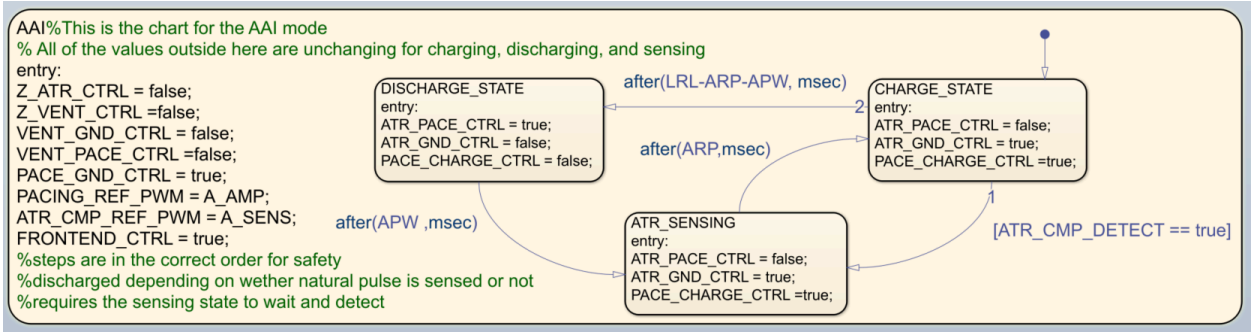
2.3.3.2 Variables

Name	Reference
t	2.2.1
ATR_CMP_DETECT	2.2.1
LRL	2.2.2
A_AMP	2.2.2
APW	2.2.2
A_SENS	2.2.2
ARP	2.2.2

2.3.3.3 Requirements

Current State	t	ATR_CMP_DETECT	Next State
CHARGE_STATE	t > LRL - ARP - APW	-	DISCHARGE_STATE
CHARGE_STATE	-	True	ATR_SENSING
DISCHARGE_STATE	t > APW	-	ATR_SENSING
ATR_SENSING	t > ARP	-	CHARGE_STATE

2.3.3.4 Stateflow diagram



2.3.3.5 State transitions

Next State — Current State	CHARGE_STATE	ATR_SENSING	DISCHARGE_STATE
CHARGE_STATE	-	ATR_CMP_DETECT == True	t > LRL - ARP - APW
ATR_SENSING	t > ARP	-	-
DISCHARGE_STATE	-	t > APW	-

2.3.3.6 Testing

Test Case*				**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**			
OFF	OFF	N/A	N/A	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass
ON	OFF	30	1	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
ON	OFF	50	1	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
ON	OFF	50	10	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

ON	OFF	60	1	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M

We need these 5 test cases, because the function of AAI involves more steps.

- The first test case is to see what happens when the atrium isn't working at all, and this is a possibility, the AAI mode should maintain the heart rate of 60BPM all on its own
- The second test case is to see when the heart is pacing far below the LRL. In this case the AAI mode should sense when the heart beats itself, wait for a beat and then send a beat when none is sensed in the required interval. The set heart rate being at 30 BPM means that this should pulse once between 2 pulses.
- The third case is where the natural BPM is slightly below the LRLm but in this case because the pulse width of the natural heart beat is still low enough where the gap between the pulses is greater than the LRL, the pacemaker will pulse once per LRL.
- The fourth case is similar to the third except the pulse width of the heart is larger, large enough to make it so that the gap between the heart pusles is enough to not trigger the pacemaker which is exactly what happens.
- The fifth test case is set to the LRL BPM, and this should not trigger the AAI, since the heart is already pacing at the LRL.

Heartview outputs for the test cases:

Report ID: AAI_TEST_1

Active Test Routine

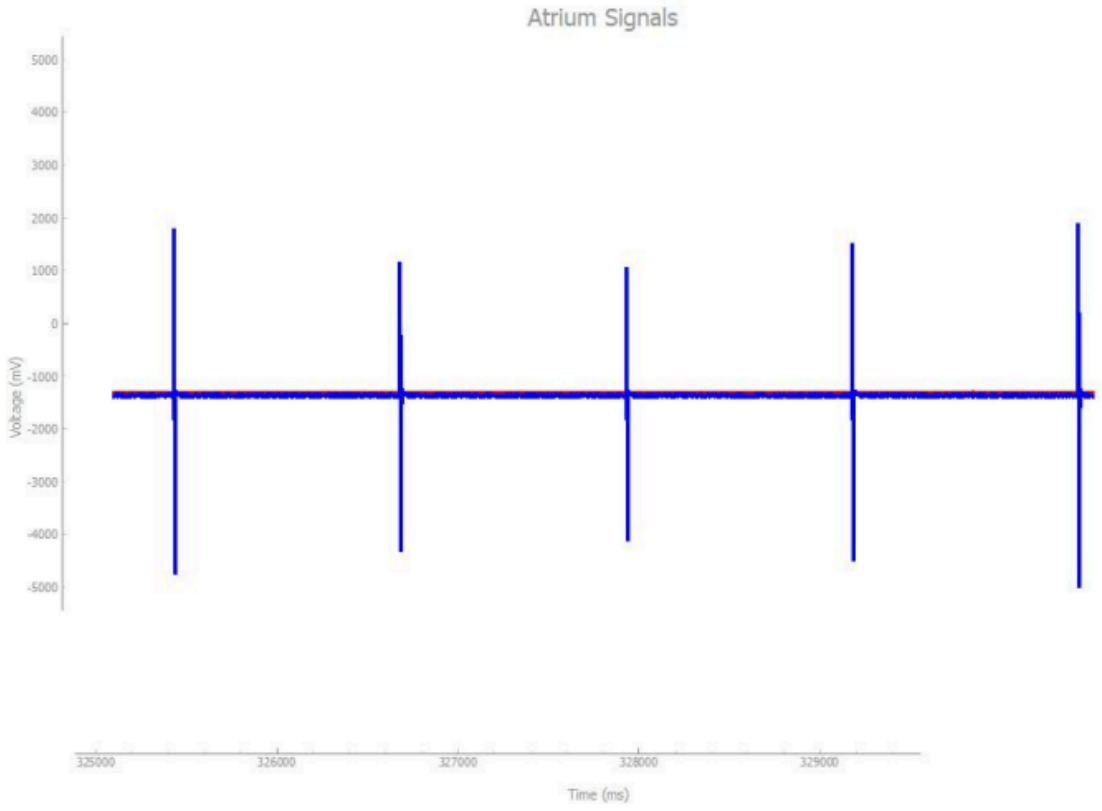
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Report ID: AAI_TEST-2

Active Test Routine

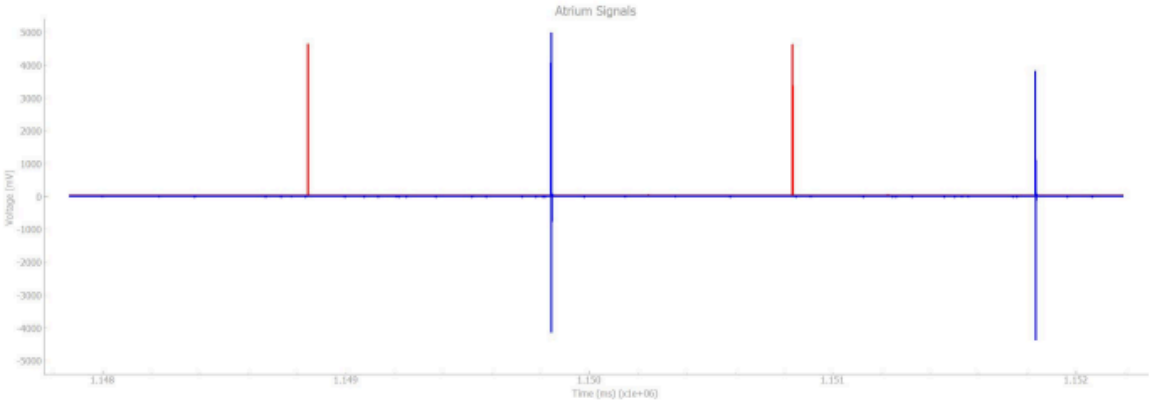
Atrium PW: 1.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Report ID: AAI_TEST-3

Active Test Routine

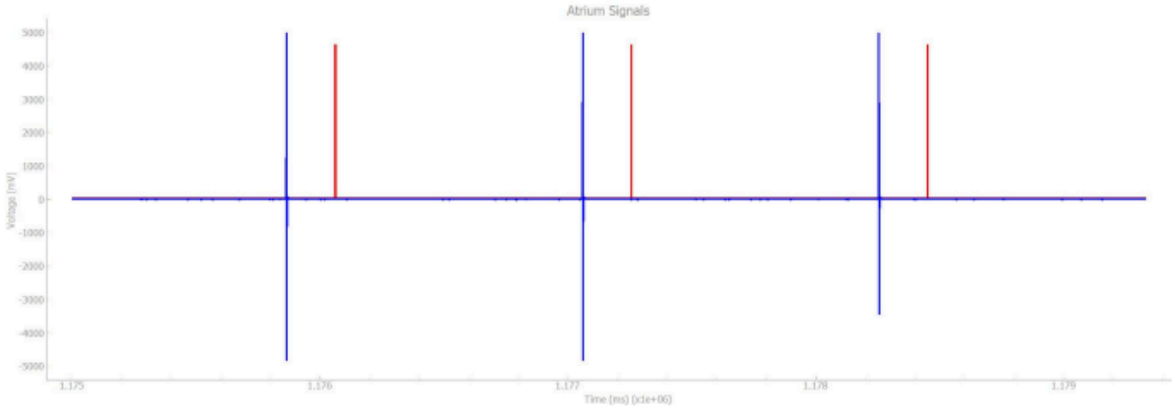
Atrium PW: 1.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 50 BPM

AV Delay: 30 ms

Atrium Plot:



Report ID: AAI_TEST_4

Active Test Routine

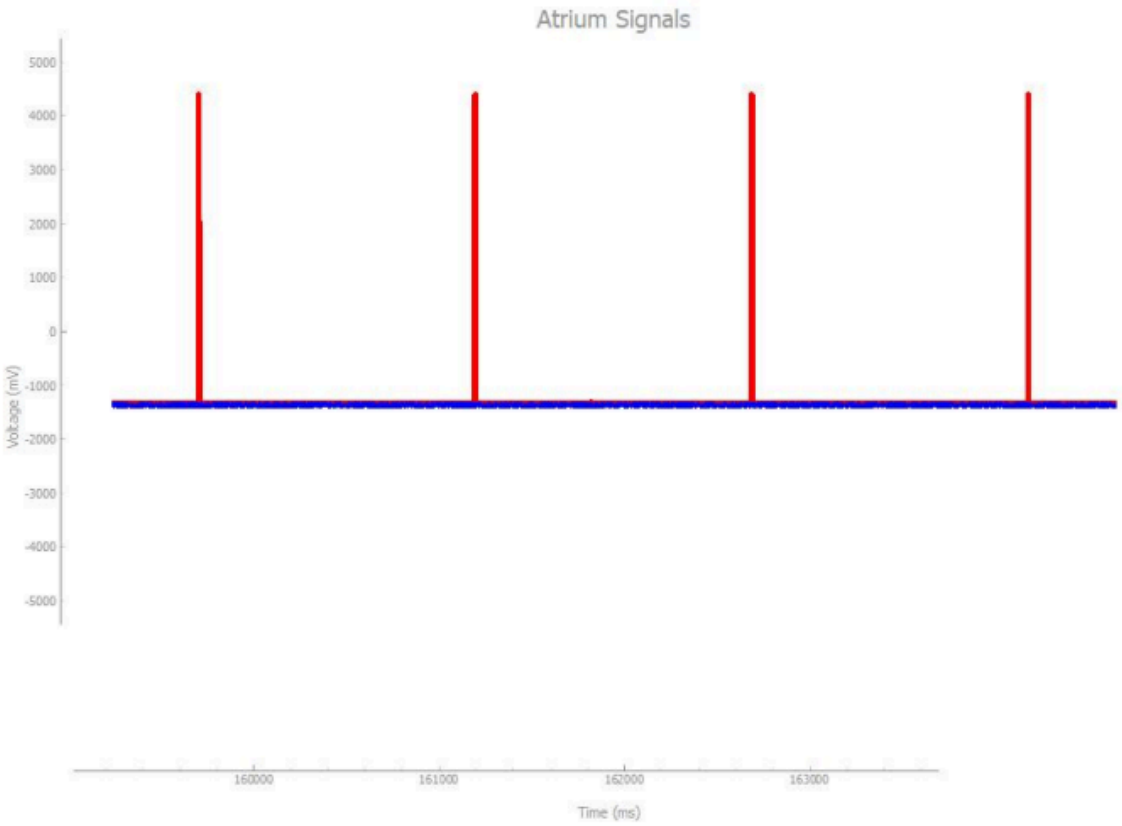
Atrium PW: 10.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 50 BPM

AV Delay: 30 ms

Atrium Plot:



Report ID: AAI_TEST_5

Active Test Routine

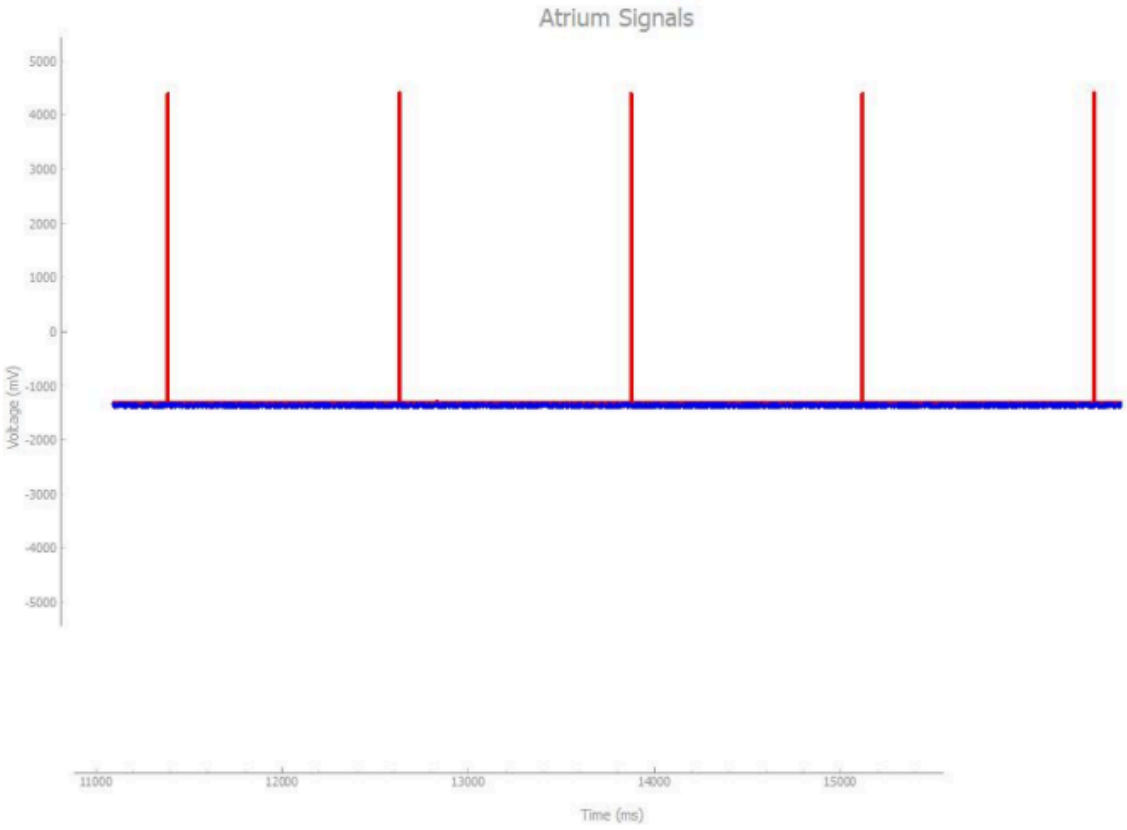
Atrium PW: 1.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 60 BPM

AV Delay: 30 ms

Atrium Plot:



2.3.3.7 Future Changes

In the future, two potential features to consider are the incorporation of hysteresis pacing and rate smoothing functions. These additions can significantly improve the performance and adaptability of the pacemaker system

Hysteresis pacing allows the pacemaker to wait more time after sensing a natural heart pulse before sending a pacing pulse. This encourages the heart to beat on its own by providing this longer delay following a sensed atrial pulse and potentially avoids unnecessary pacing. If the natural heart pulse does not resume within this extended interval, the pacemaker will step in and deliver the pulse. Incorporating hysteresis pacing additionally reduces the pacing burden, can preserve battery life, and promotes a more natural heart rhythm.

To incorporate this feature, we would create a hysteresis rate limit (HRL) which would be slightly below the lower rate limit (LRL) to give the heart more time to pulse on its own. Then by adding logic to detect a sensed atrial pulse and extend the pacing interval beyond the normal pacing period if hysteresis is enabled and logic that ensures the pacemaker remains in charge state and does not deliver a pulse if an arterial pulse is detected.

Rate smoothing helps limit sudden changes in pacing rates. By setting limits on how much the pacing rate can increase or decrease it prevents the pacing rate from changing too abruptly. Abrupt changes in heart rate can result in side effects such as dizziness and by adding rate smoothing we can help to avoid this.

To incorporate rate smoothing, we can introduce two new parameters for rate smoothing up and rate smoothing down and setting them as a percentage of the previous cardiac cycle interval. Then we can program the pacemaker to adjust the pacing rate to stay within these limits whenever the detected intrinsic rate changes ensuring smooth transitions.

The incorporation of hysteresis pacing and rate smoothing would enhance the pacemaker’s ability to promote natural heart functions and provide more adaptive responses, providing an overall better experience for the user.

2.3.4 VVI

2.3.4.1 Overview

This is the mode to first sense the ventricle and in response to the natural ventricular pulse, inhibit the pacemaker pulse, or in lack of ventricular pulse let the pacemaker pulse the heart.(Ventricular Pacing, Ventricular Sensing, Inhibiting)

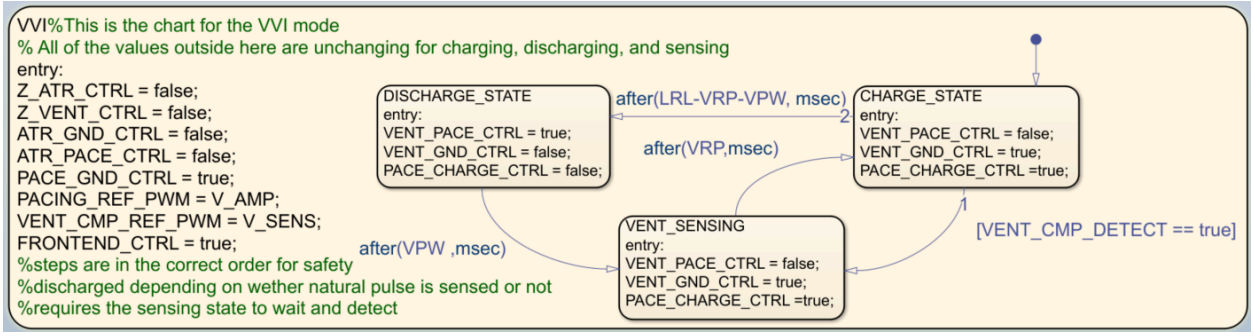
2.3.4.2 Variables

Name	Reference
t	2.2.1
VENT_CMP_DETECT	2.2.1
LRL	2.2.2
V_AMP	2.2.2
VPW	2.2.2
V_SENS	2.2.2
VRP	2.2.2

2.3.4.3 Requirements

Current State	t	VENT_CMP_DETECT	Next State
CHARGE_STATE	t > LRL - VRP - VPW	-	DISCHARGE_STATE
CHARGE_STATE	-	True	VENT_SENSING
DISCHARGE_STATE	t > VPW	-	VENT_SENSING
VENT_SENSING	t > VRP	-	CHARGE_STATE

2.3.4.4 Stateflow diagram



2.3.4.5 State transitions

Next State — Current State	CHARGE_STATE	VENT_SENSING	DISCHARGE_STATE
CHARGE_STATE	-	VENT_CMP_DETECT == True	t > LRL - VRP - VPW
VENT_SENSING	t > VRP	-	-
DISCHARGE_STATE	-	t > VPW	-

2.3.4.6 Testing

Test Case*				**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**			
OFF	OFF	N/A	N/A	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass
OFF	ON	30	1	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Ventricle once per LRL	Pass
OFF	ON	50	1	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Ventricle once per LRL	Pass
OFF	ON	50	10	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

OFF	ON	60	1	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M

We need these 5 test cases, because the function of VVI involves more steps.

- The first test case is to see what happens when the ventricle isn't working at all, and this is a possibility, the VVI mode should maintain the heart rate of 60BPM all on its own
- The second test case is to see when the heart is pacing far below the LRL. In this case the VVI mode should sense when the heart beats itself, wait for a beat and then send a beat when none is sensed in the required interval. The set heart rate being at 30 BPM means that this should pulse once between 2 pulses.
- The third case is where the natural BPM is slightly below the LRLm but in this case because the pulse width of the natural heart beat is still low enough where the gap between the pulses is greater than the LRL, the pacemaker will pulse once per LRL.
- The fourth case is similar to the third except the pulse width of the heart is larger, large enough to make it so that the gap between the heart pusles is enough to not trigger the pacemaker which is exactly what happens.
- The fifth test case is set to the LRL BPM, and this should not trigger the VVI, since the heart is already pacing at the LRL.

Heartview outputs for the test cases:

Report ID: VVI_TEST_1

Active Test Routine

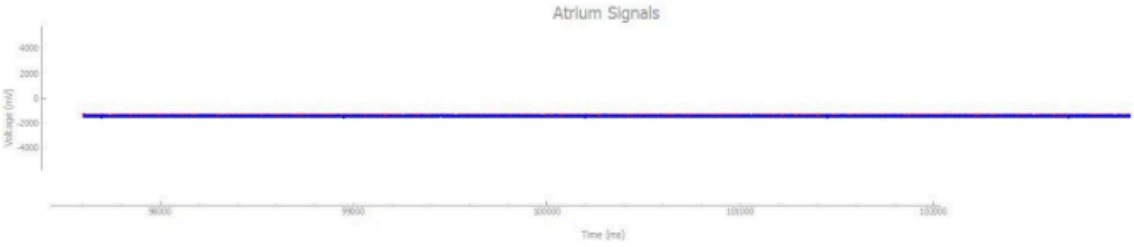
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

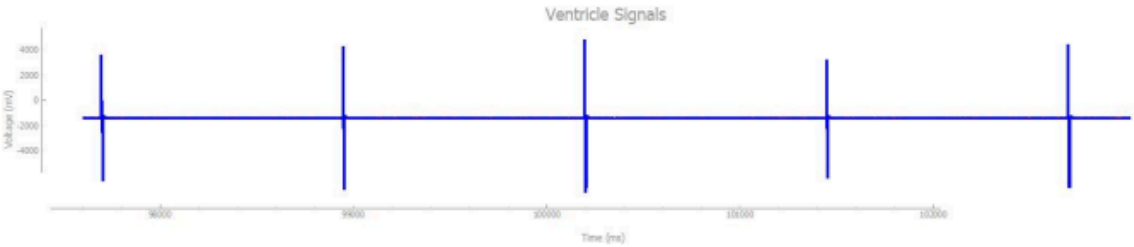
Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Ventricle Plot:



Report ID: VVI_TEST_2

Active Test Routine

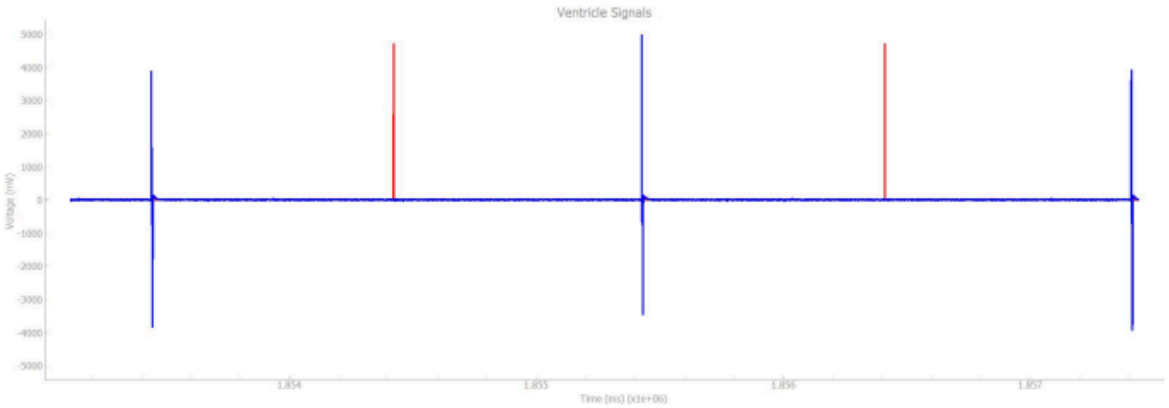
Atrium PW: 0.0 ms

Ventricular PW: 1.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Ventricle Plot:



Report ID: VVI_TEST_3

Active Test Routine

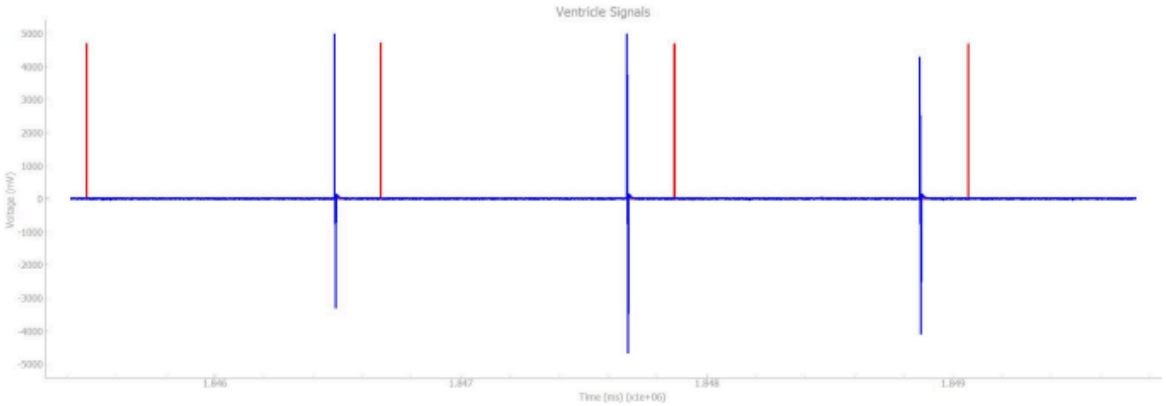
Atrium PW: 0.0 ms

Ventricular PW: 1.0 ms

Heart Rate: 50 BPM

AV Delay: 30 ms

Ventricle Plot:



Report ID: VVI_TEST_4

Active Test Routine

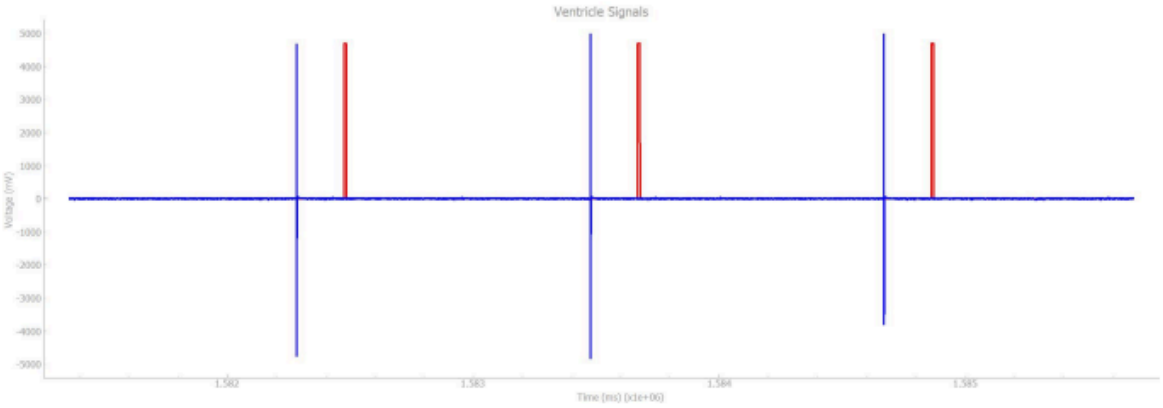
Atrium PW: 0.0 ms

Ventricular PW: 10.0 ms

Heart Rate: 50 BPM

AV Delay: 30 ms

Ventricle Plot:



Report ID: VVI_TEST_5

Active Test Routine

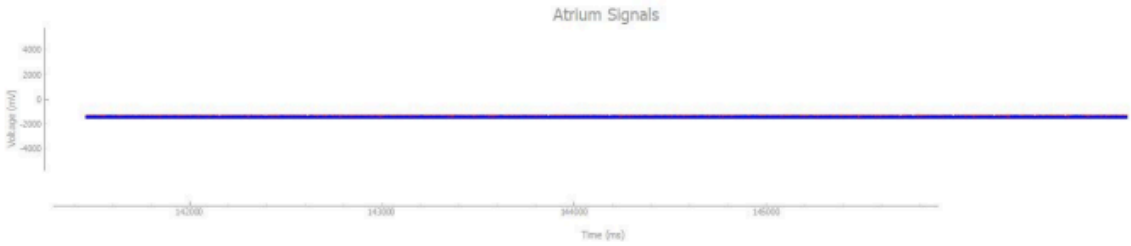
Atrium PW: 0.0 ms

Ventricular PW: 1.0 ms

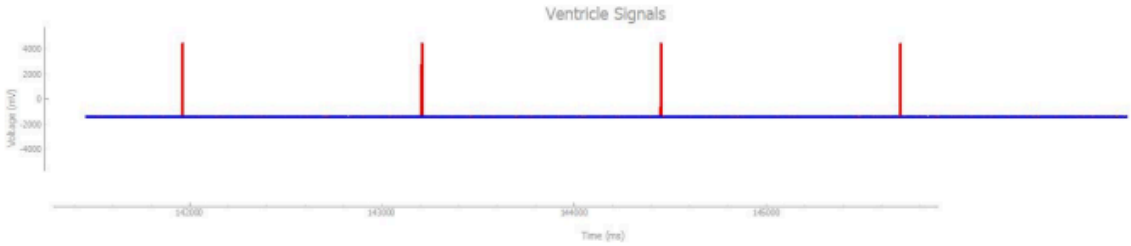
Heart Rate: 60 BPM

AV Delay: 30 ms

Atrium Plot:



Ventricle Plot:



2.3.4.7 Future Changes

In the future, two potential features to consider are the incorporation of hysteresis pacing and rate smoothing functions. These additions can significantly improve the performance and adaptability of the pacemaker system

Hysteresis pacing allows the pacemaker to wait more time after sensing a natural heart pulse before sending a pacing pulse. This encourages the heart to beat on its own by providing this longer delay following a sensed ventricular pulse and potentially avoids unnecessary pacing. If the natural heart pulse does not resume within this extended interval, the pacemaker will step in and deliver the pulse. Incorporating hysteresis pacing additionally reduces the pacing burden, can preserve battery life, and promotes a more natural heart rhythm.

To incorporate this feature, we would create a hysteresis rate limit (HRL) which would be slightly below the lower rate limit (LRL) to give the heart more time to pulse on its own. Then by adding logic to detect a sensed ventricular pulse and extend the pacing interval beyond the normal pacing period if hysteresis is enabled and logic that ensures the pacemaker remains in charge state and does not deliver a pulse if a ventricular pulse is detected.

Rate smoothing helps limit sudden changes in pacing rates. By setting limits on how much the pacing rate can increase or decrease it prevents the pacing rate from changing too abruptly. Abrupt changes in heart rate can result in side effects such as dizziness and by adding rate smoothing we can help to avoid this.

To incorporate rate smoothing, we can introduce two new parameters for rate smoothing up and rate smoothing down and setting them as a percentage of the previous cardiac cycle interval. Then we can program the pacemaker to adjust the pacing rate to stay within these limits whenever the detected intrinsic rate changes ensuring smooth transitions.

The incorporation of hysteresis pacing and rate smoothing would enhance the pacemaker's ability to promote natural heart functions and provide more adaptive responses, providing an overall better experience for the user.

2.3.5 Rate Adaptive Pacing

Rate adaptive pacing is a feature that we implemented that takes the built in accelerometer data from the board, and simulates the shaking/moving that a normal person would have in their daily lives. These values are dependent on how much the board is accelerating which allows us to map specific acceleration values to activity levels which is used to determine the rate at which the pacemaker pulses the heart. While the original functionality of different modes is preserved, the rate adaptive modes allow for faster response if required, up to the Upper Rate Limit (URL).

The 4 new modes that incorporate rate adaptive pacing are described as follows:

2.3.5.1 AOOR

2.3.5.1.1 Overview

This mode is meant to pace the atrium regardless of the current state of the atrium. However, the additional feature is rate adaptivity that allows the pacing rate to increase with an increase in activity, up to URL.

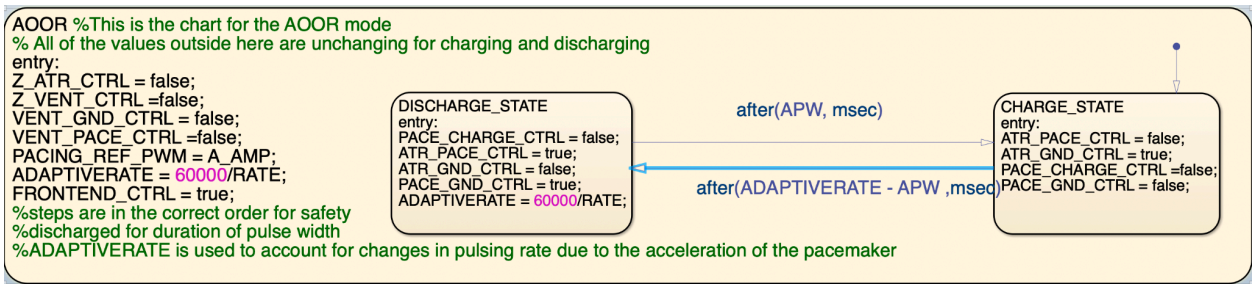
2.3.5.1.2 Variables

Name	Reference
t	2.2.1
A_AMP	2.2.2
APW	2.2.2
ADAPTIVERATE	Local variable
RATE	2.2.2

2.3.5.1.3 Requirements

Current state	t value	Next state
Charging	t > ADAPTIVERATE - APW	Discharging
Discharging	t > APW	Charging

2.3.5.1.4 Stateflow diagram

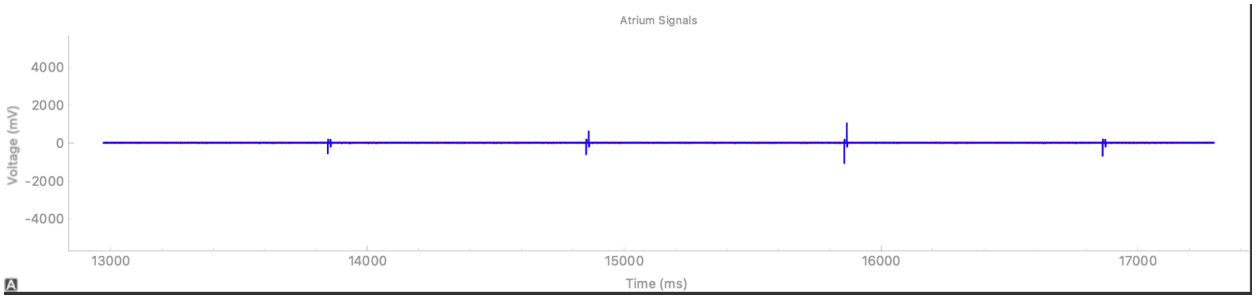


2.3.5.1.5 Testing

Test Case			Expected Result	Actual Result	Status
Natural Atrium	**Natural Ventricle**	**Pacemaker Shaking**			
OFF	OFF	Not shaking	Pacemaker pulsing Atrium at LRL	Pacemaker pulsing Atrium at LRL	Pass
OFF	OFF	Moderate Shaking	Pacemaker pulsing Atrium at ADAPTIVERATE	Pacemaker pulsing Atrium at ADAPTIVERATE	Pass
OFF	OFF	High Shaking	Pacemaker pulsing Atrium at URL	Pacemaker pulsing Atrium at URL	Pass

*The LRL is set to 60P/M
*The URL is set to 120P/M

The AOOR mode is meant to be very similar in functionality to the AOO mode where it is expected to pace the heart at LRL if no natural pulse is detected but increase the pulsing rate if the absence of natural pulse is accompanied by a lot of movement i.e. high activity levels. The pulsing rate can be increased up to the URL value.



AOOR heart view output when heart is not pulsing and the person is moderately moving

2.3.5.2 VOOR

2.3.5.2.1 Overview

This mode is meant to pace the ventricle regardless of the current state of the ventricle. The additional feature is rate adaptivity that allows the pacing rate to increase with an increase in activity, up to the URL.

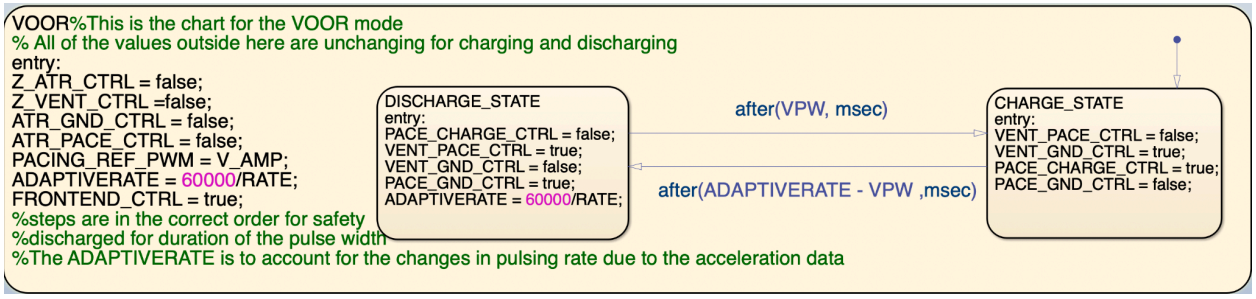
2.3.5.2.2 Variables

Name	Reference
t	2.2.1
V_AMP	2.2.2
VPW	2.2.2
ADAPTIVERATE	Local variable
RATE	2.2.2

2.3.5.2.3 Requirements

Current state	t value	Next state
Charging	t > ADAPTIVERATE - VPW	Discharging
Discharging	t > VPW	Charging

2.3.5.2.4 Stateflow diagram

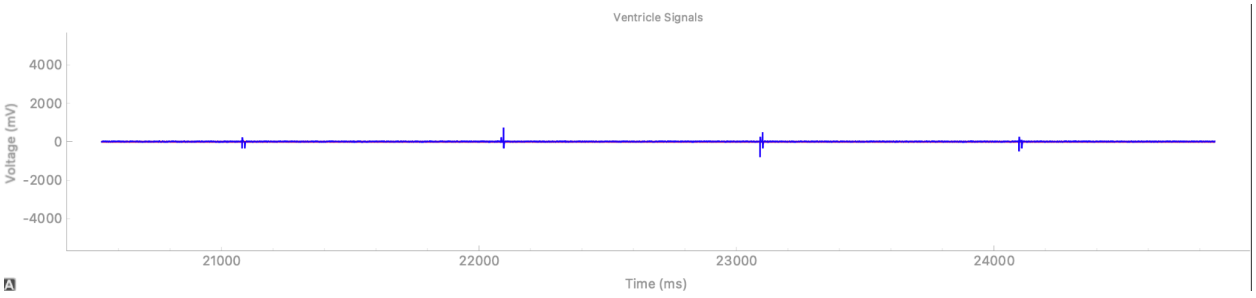


2.3.5.2.5 Testing

Test Case			Expected Result	Actual Result	Status
Natural Atrium	**Natural Ventricle**	**Pacemaker Shaking**			
OFF	OFF	Not shaking	Pacemaker pulsing Ventricle at LRL	Pacemaker pulsing Ventricle at LRL	Pass
OFF	OFF	Moderate Shaking	Pacemaker pulsing Ventricle at ADAPTIVERATE	Pacemaker pulsing Ventricle at ADAPTIVERATE	Pass
OFF	OFF	High Shaking	Pacemaker pulsing Ventricle at URL	Pacemaker pulsing Ventricle at URL	Pass

*The LRL is set to 60P/M
*The URL is set to 120P/M

The VOOR mode is meant to be very similar in functionality to the VOO mode where it is expected to pace the heart at LRL if no natural pulse is detected but increase the pulsing rate if the absence of natural pulse is accompanied by a lot of movement i.e. high activity levels. The pulsing rate can be increased up to the URL value.



Testing for VOOR mode when no natural ventricular pulse is there.

2.3.5.3 AAIR

2.3.5.3.1 Overview

This is the mode to first sense the atrium and in response to the natural atrium pulse, inhibit the pacemaker pulse, or in lack of atrial pulse let the pacemaker pulse the heart. (Atrial Pacing, Atrial Sensing, Inhibiting). The additional functionality includes the ability to dynamically change the pacing rate in response to the accelerometer values.

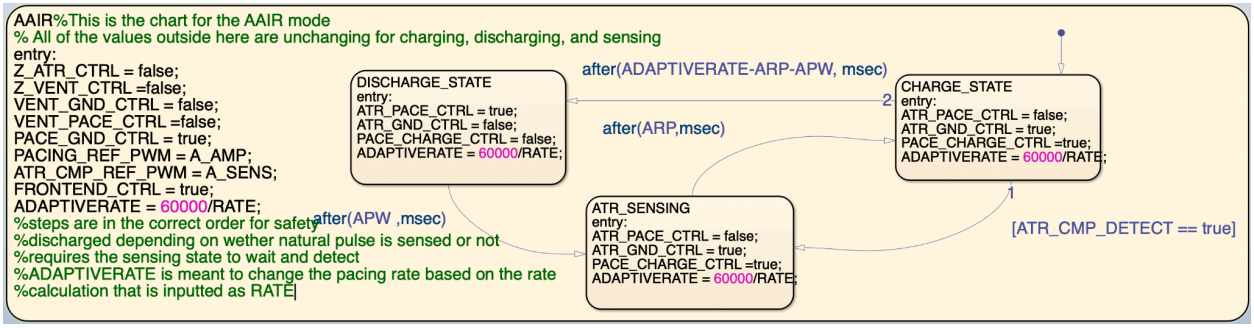
2.3.5.3.2 Variables

Name	Reference
t	2.2.1
ATR_CMP_DETECT	2.2.1
LRL	2.2.2
A_AMP	2.2.2
APW	2.2.2
A_SENS	2.2.2
ARP	2.2.2
ADAPTIVERATE	Local variable
RATE	2.2.2

2.3.5.3.3 Requirements

Current State	t	ATR_CMP_DETECT	Next State
CHARGE_STATE	t > ADAPTIVERATE - ARP - APW	-	DISCHARGE_STATE
CHARGE_STATE	-	True	ATR_SENSING
DISCHARGE_STATE	t > APW	-	ATR_SENSING
ATR_SENSING	t > ARP	-	CHARGE_STATE

2.3.5.3.4 Stateflow diagram



2.3.5.3.5 State transitions

Next State — **Current State**	**CHARGE_STATE**	**ATR_SENSING**	**DISCHARGE_STATE**
CHARGE_STATE	-	ATR_CMP_DETECT == True	t > ADAPTIVERATE - ARP - APW
ATR_SENSING	t > ARP	-	-
DISCHARGE_STATE	-	t > APW	-

2.3.5.3.6 Testing

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacemaker shaking**			
OFF	OFF	N/A	N/A	No shaking	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass
ON	OFF	30	1	No shaking	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
ON	OFF	50	1	No shaking	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass

ON	OFF	50	10	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
ON	OFF	60	1	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	Moderate shaking	Pacemaker Pulsing Atrium at ADAPTIVERATE	Pacemaker Pulsing Atrium at ADAPTIVERATE	Pass
ON	OFF	30	1	Moderate shaking	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pass
ON	OFF	50	1	Moderate shaking	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pass
ON	OFF	50	10	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
ON	OFF	60	1	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	High shaking	Pacemaker Pulsing Atrium at URL	Pacemaker Pulsing Atrium at URL	Pass
ON	OFF	30	1	High shaking	Pacemaker Pulsing Atrium once per URL	Pacemaker Pulsing Atrium once per URL	Pass
ON	OFF	50	1	High shaking	Pacemaker Pulsing Atrium once per URL	Pacemaker Pulsing Atrium once per URL	Pass
ON	OFF	50	10	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
ON	OFF	60	1	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

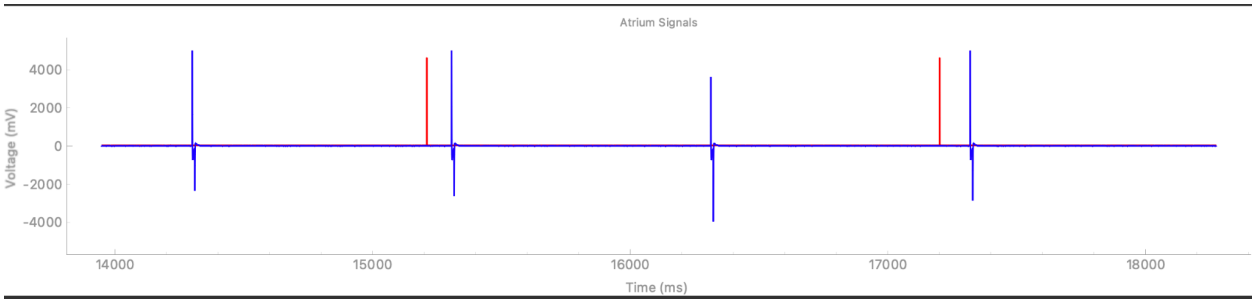
*The LRL is set to 60P/M
*The URL is set to 120P/M

*Moderate shaking is defined as any values from the accelerometer that increase the pacing rate above LRL but don't cause it to reach the URL.
*High shaking is defined as any values from the accelerometer that cause the pacing rate to reach the URL.

We need these 3 activity scenarios which amount to a total of 15 test cases, because the function of AAIR involves more steps:

- The first set of 5 test cases are to see what happens when there is no movement at all i.e. the accelerometer values are zero. However, different natural atrium heart beats are possible which are captured in these 5 test cases. It is worth noting that the results of this test set correspond to that of AAI mode.
- The second set of 5 test cases are to see how the pacemaker responds if there is moderate movement. The mode works in a similar fashion to AAI but is able to change the pacing rate in response to the accelerometer input. The pacing rate stays within [LRL, URL].
- The third set of 5 test cases are to see how the pacemaker responds to a high amount of movement which would imply that the user is engaging in heavy exercise. The mode works in similar fashion to AAI but is able to change the pacing rate to the URL in response to the high values from the accelerometer.
- For a detailed explanation of the working of the AAI mode, please refer to **section 2.3.3**

Heartview outputs for the test case:



Heartview output for AAIR mode when heart is pulsing and the user is undergoing moderate movement

2.3.5.4 VVIR

2.3.5.4.1 Overview

This is the mode to first sense the ventricle and in response to the natural ventricle pulse, inhibit the pacemaker pulse, or in lack of ventricle pulse let the pacemaker pulse the heart. (Ventricular Pacing, Ventricular Sensing, Inhibiting). The additional functionality includes the ability to dynamically change the pacing rate in response to the accelerometer values.

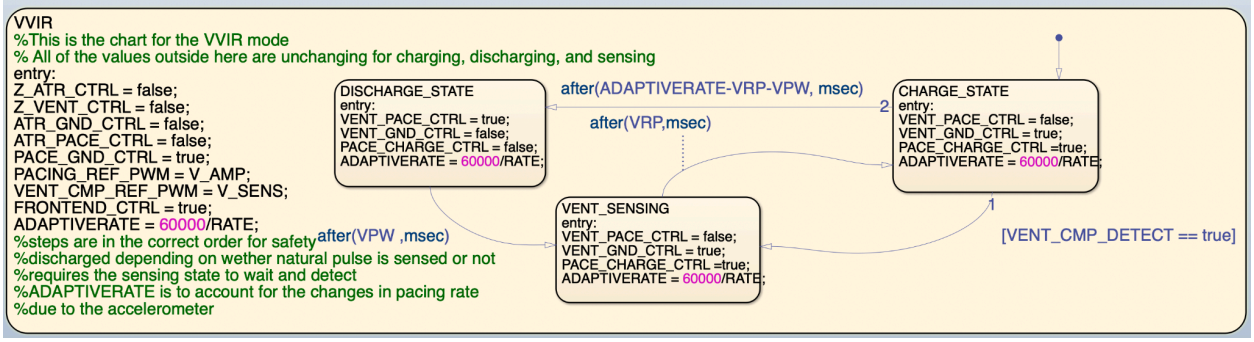
2.3.5.4.2 Variables

Name	Reference
t	2.2.1
VENT_CMP_DETECT	2.2.1
LRL	2.2.2
V_AMP	2.2.2
VPW	2.2.2
V_SENS	2.2.2
VRP	2.2.2
ADAPTIVERATE	Local variable
RATE	2.2.2

2.3.5.4.3 Requirements

Current State	t	VENT_CMP_DETECT	Next State
CHARGE_STATE	t > ADAPTIVERATE - VRP - VPW	-	DISCHARGE_STATE
CHARGE_STATE	-	True	VENT_SENSING
DISCHARGE_STATE	t > VPW	-	VENT_SENSING
VENT_SENSING	t > VRP	-	CHARGE_STATE

2.3.5.4.4 Stateflow diagram



2.3.5.4.5 State transition table

Next State — Current State	CHARGE_STATE	VENT_SENSING	DISCHARGE_STATE
CHARGE_STATE	-	VENT_CMP_DETECT == True	t > ADAPTIVERATE - VRP - VPW
VENT_SENSING	t > VRP	-	-
DISCHARGE_STATE	-	t > VPW	-

2.3.5.4.6 Testing

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacemaker shaking**			
OFF	OFF	N/A	N/A	No shaking	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass
OFF	ON	30	1	No shaking	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Ventricle once per LRL	Pass
OFF	ON	50	1	No shaking	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
OFF	ON	50	10	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	1	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	Moderate shaking	Pacemaker Pulsing Ventricle at ADAPTIVERATE	Pacemaker Pulsing Ventricle at ADAPTIVERATE	Pass

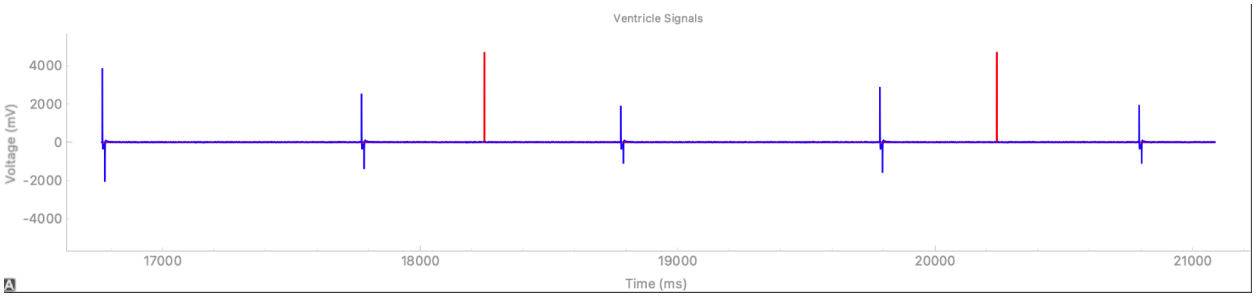
OFF	ON	30	1	Moderate shaking	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pass
OFF	ON	50	1	Moderate shaking	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pass
OFF	ON	50	10	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	1	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	High shaking	Pacemaker Pulsing Ventricle at URL	Pacemaker Pulsing Ventricle at URL	Pass
OFF	ON	30	1	High shaking	Pacemaker Pulsing Ventricle once per URL	Pacemaker Pulsing Ventricle once per URL	Pass
OFF	ON	50	1	High shaking	Pacemaker Pulsing Ventricle once per URL	Pacemaker Pulsing Ventricle once per URL	Pass
OFF	ON	50	10	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	1	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M
*The URL is set to 120P/M
*Moderate shaking is defined as any values from the accelerometer that increase the pacing rate above LRL but don’t cause it to reach the URL.
*High shaking is defined as any values from the accelerometer that cause the pacing rate to reach the URL.

We need these 3 activity scenarios which amount to a total of 15 test cases, because the function of VVIR involves more steps:

- The first set of 5 test cases are to see what happens when there is no movement at all i.e. the accelerometer values are zero. However, different natural ventricle heart beats are possible which are captured in these 5 test cases. It is worth noting that the results of this test set correspond to that of VVI mode.
- The second set of 5 test cases are to see how the pacemaker responds if there is moderate movement. The mode works in a similar fashion to VVI but is able to change the pacing rate in response to the accelerometer input. The pacing rate stays within [LRL, URL].
- The third set of 5 test cases are to see how the pacemaker responds to a high amount of movement which would imply that the user is engaging in heavy exercise. The mode works in similar fashion to VVI but is able to change the pacing rate to the URL in response to the high values from the accelerometer.
- For a detailed explanation of the working of the VVI mode, please refer to **section 2.3.3**

Heartview outputs for the test case:



Heartview output for VVIR when the Ventricle is pulsing and the user is engaged in moderate activity.

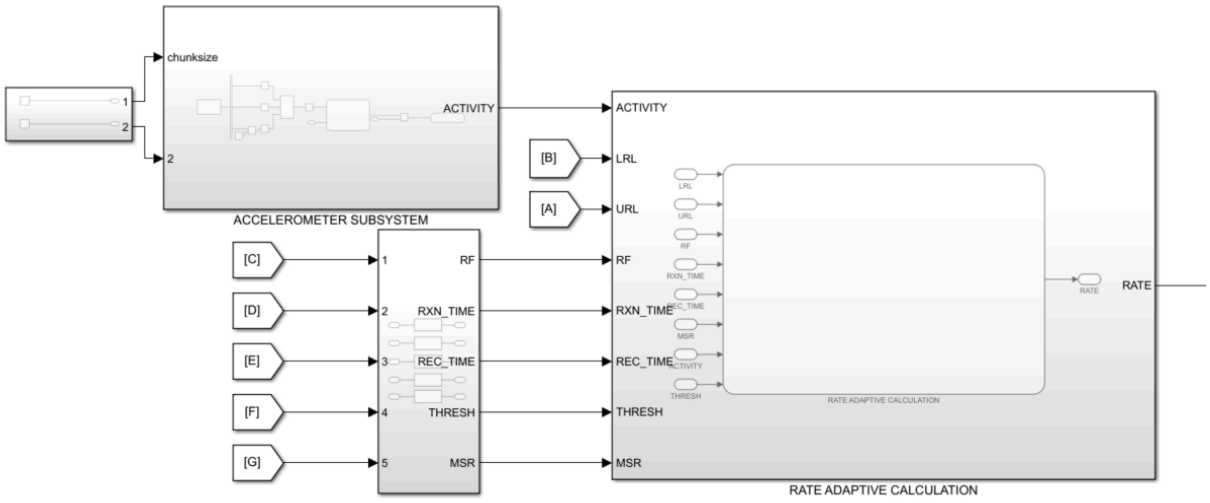
2.3.5.5 Rate Calculations

2.3.5.5.1 Overview

There are 2 parts to the calculations. It starts inside of the accelerometer subsystem, where we take the accelerometer data, split it into its 3 components x,y,z and then find the magnitude of the overall vector by squaring, adding and square rooting. Before we put the z data in, we do 1-z to take gravity into account. We then input it into the block where we calculate a chunked average to filter out the noise. The values that we used for chunksize are arbitrary, and can be adjusted to fit different intervals, we just found that 10 was a good mix to update frequently and avoid excess noise. Then we finally multiplied that value by a gain of 40, to make it easier to work with, this was also found through trial and error, and we found that 40 is what worked best for us.

The second part is where we input that activity level into a subsystem that we use to convert it into a rate that we can use. This works by taking into account other factors such as a reaction

and recovery time as well as a response factor and a maximum sensing rate. We also make sure that the rate stays between the lower and upper limit for the heart rate. There is also a threshold level that is adjustable based on the user input.



This is an image of the rate adaptive calculation portion of the simulink

The images and the subsystems were broken down in more detail above in section 2.1.2

2.3.5.5.2 Requirements

Subsystem	**Input data**	**Process**	**Output data**
ACCELEROMETER SUBSYSTEM	Accelerometer data, chunksize, factor	Convert raw data into acceleration magnitude	ACTIVITY = Average accelerometer data over 10 samples with a gain of 40.
RATE ADAPTIVE CALCULATION	LRL, URL, RF, REACTION TIME, RECOVERY TIME, MSR, ACTIVITYIN=ACTIVITY, THRESH	Please refer to **2.3.5.5.2.1 Rate Adaptive Calculation Stateflow**	RATE = pacing rate as a function of the input variables

2.3.5.5.2.1 Rate Adaptive Calculation State Transition Table

Next State — **Current State**	**START**	**SLOPE_CALCULTION**	**RATE_CALCULATION**
START	-	instantly	-

SLOPE_CALCULATION	-	-	t > 1ms
RATE_CALCULATION	-	t > 100ms	-

2.3.5.5.3 Testing

Refer to sections 2.3.5.1.5, 2.3.5.2.5, 2.3.5.3.6, and 2.3.5.4.6 for all the testing done with the adaptive pacing within the 4 different modes that employ it (AOOR, VOOR, AAIR, VVIR).

2.3.5.5.4 Future Changes

For the rate adaptive pacing, we don't anticipate any future changes being necessary since it is complete as is. A future change related to this is implementing DDD and DDDR which is a mode that involves rate adaptive pacing.

2.4 Hardware Hiding

2.4.1 Overview

Hardware hiding is a key feature in Simulink modeling, especially when dealing with complex systems involving Stateflow logic. It abstracts the model's logic from the underlying hardware, providing several benefits. First, it allows the model to remain versatile and adaptable, ensuring that the system logic is not tied to any specific hardware configuration. This is particularly useful during prototyping and development phases, as it expedites the process by allowing engineers to focus on refining the control logic without being constrained by hardware limitations or potential hardware changes. Additionally, hardware hiding simplifies future upgrades and modifications. If the hardware is changed—whether due to performance upgrades, availability issues, or cost considerations—the model remains largely unaffected. Engineers can make quick adjustments to the input/output configurations without altering the core logic, enhancing flexibility and enabling quicker adaptation to evolving project needs. As shown in section 2.1.2, where images of our hardware hiding implementation are provided, we have employed this abstraction for both input and output systems. This ensures that any changes to the hardware setup require minimal modification to the model, promoting modularity and reusability across different platforms or projects. By adopting this approach, we ensured a more scalable and future-proof design.

2.4.2 Pacing

2.4.2.1 Description

The pacing states in each of the modes are the discharge states where it discharges the capacitor into the heart. It regulates the control signals and the pacing reference pulse width modulation in the pacing circuitry. They are designed to pace the heart for the pulse width that is predefined as a control variable.

2.4.2.2 Variables

Name	Reference
t	2.2.1
APW	2.2.2
VPW	2.2.2
A_AMP	2.2.2
V_AMP	2.2.2
Mode	2.2.2

2.4.2.3 Requirements

Variable	Current Value	t value	New Value
ATR_PACE_CTRL	false	t > LRL - APW	true
ATR_PACE_CTRL	true	t > APW	false
ATR_GND_CTRL	true	t > LRL - APW	false
ATR_GND_CTRL	false	t > APW	true
PACE_CHARGE_CTRL	true	t > LRL - VPW OR t > LRL - APW	false
PACE_CHARGE_CTRL	false	t > VPW OR t > APW	true
PACE_GND_CTRL	false	t > LRL - VPW OR t > LRL - APW	true
PACE_GND_CTRL	true	t > VPW OR t > APW	false
VENT_PACE_CTRL	false	t > LRL - VPW	true
VENT_PACE_CTRL	true	t > VPW	false
VENT_GND_CTRL	true	t > LRL - VPW	false
VENT_GND_CTRL	false	t > VPW	true

2.4.2.4 Testing

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacing Mode**			
OFF	OFF	N/A	N/A	AOO	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass

OFF	OFF	N/A	N/A	VOO	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass

*The LRL is set to 60P/M

These test cases demonstrate the functionality of AOO and VOO as they should both pulse at LRL when no natural pulse is detected

Heartview outputs for the test cases:

Report ID: PACING_TEST_1

Active Test Routine

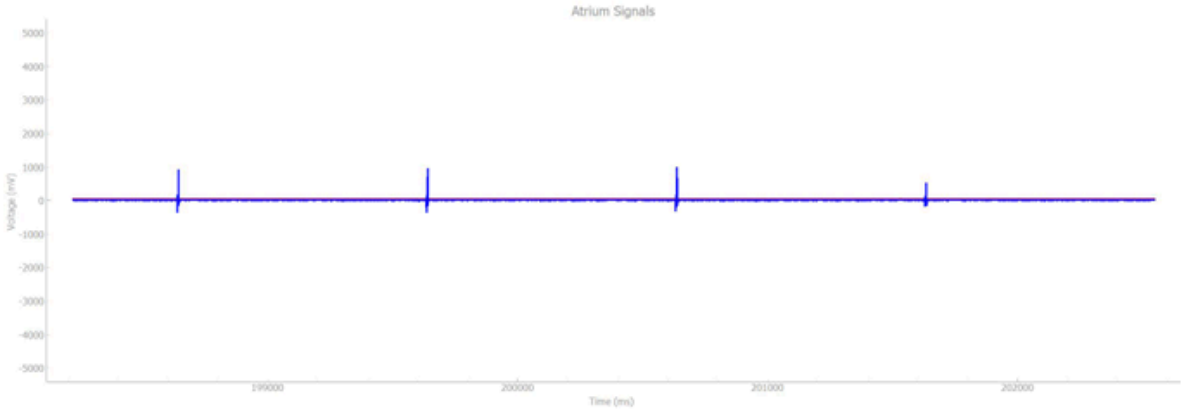
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Report ID: PACING_TEST_2

Active Test Routine

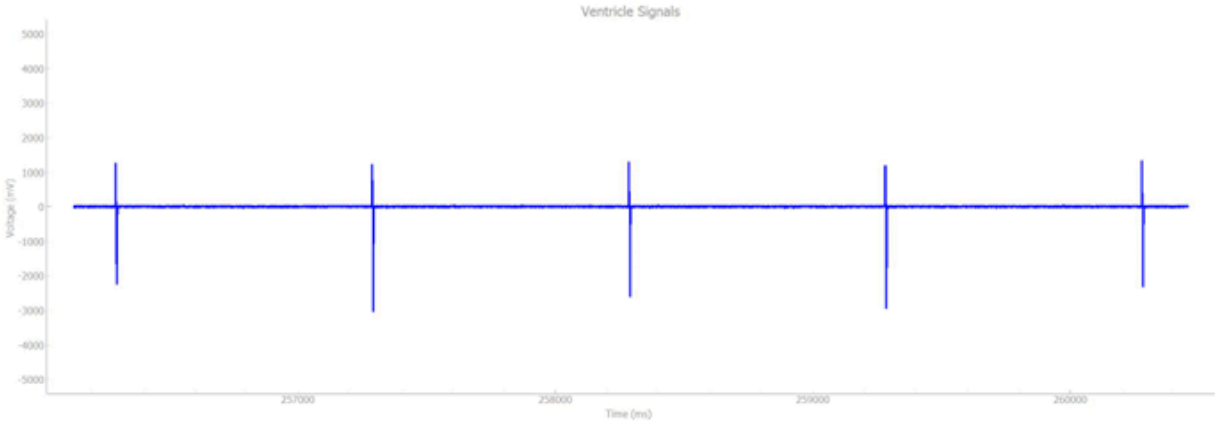
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Ventricle Plot:



2.4.2.5 Future Changes

In the future, we may need to pace both the ventricle and atrium at the same time, so some adjustments might be needed for combining both pacing into one state.

2.4.3 Charging/Sensing

2.4.2.1 Description

The charging/sensing states in each of the modes are the same, and they both charge the capacitor that is responsible for pacing. This acts as both a buffer state when the natural atrial/ventricular pulses are detected in AAI/VVI and then charging for all the modes.

2.4.2.2 Variables

Name	Reference
t	2.2.1
ATR_CMP_DETECT	2.2.1
VENT_CMP_DETECT	2.2.1
A_SENS	2.2.2
V_SENS	2.2.2
Mode	2.2.2

2.4.2.3 Requirements

Variable	Current Value	t value	New Value
ATR_PACE_CTRL	false	t > LRL - ARP - APW OR ATR_CMP_DETECT == true	True OR False
ATR_PACE_CTRL	true	t > APW	false
ATR_GND_CTRL	true	t > LRL - ARP - APW OR ATR_CMP_DETECT == true	False OR True
ATR_GND_CTRL	false	t > APW	true
PACE_CHARGE_CTRL	true	t > LRL - ARP - APW OR	False OR

		ATR_CMP_DETECT == true t > LRL - VRP - VPW OR VENT_CMP_DETECT == true	True False OR True
PACE_CHARGE_CTRL	false	t > VPW	true
VENT_PACE_CTRL	false	t > LRL - VRP - VPW OR VENT_CMP_DETECT == true	True OR False
VENT_PACE_CTRL	true	t > VPW	false
VENT_GND_CTRL	true	t > LRL - VRP - VPW OR VENT_CMP_DETECT == true	False OR True
VENT_GND_CTRL	false	t > VPW	true

2.4.2.4 Testing

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacing Mode**			
ON	OFF	60	10.0	AAI	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	10.0	VVI	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M
These test cases demonstrate the functionality of AII and VVI as they should not pulse when the BPM is greater than or equal to the LRL.
Heartview outputs for the test cases:

Report ID: CHARGING_TEST_1

Active Test Routine

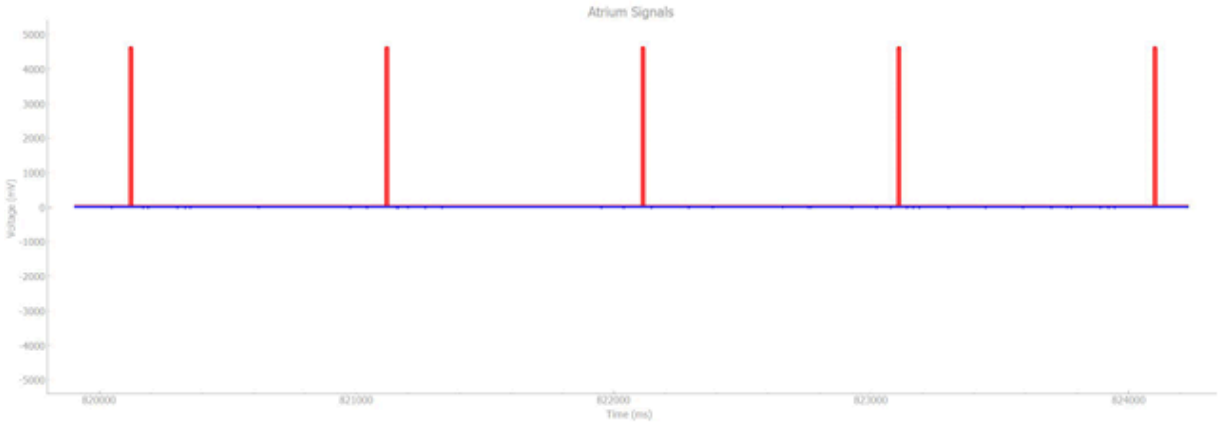
Atrium PW: 10.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 60 BPM

AV Delay: 30 ms

Atrium Plot:



Report ID: CHARGING_TEST_2

Active Test Routine

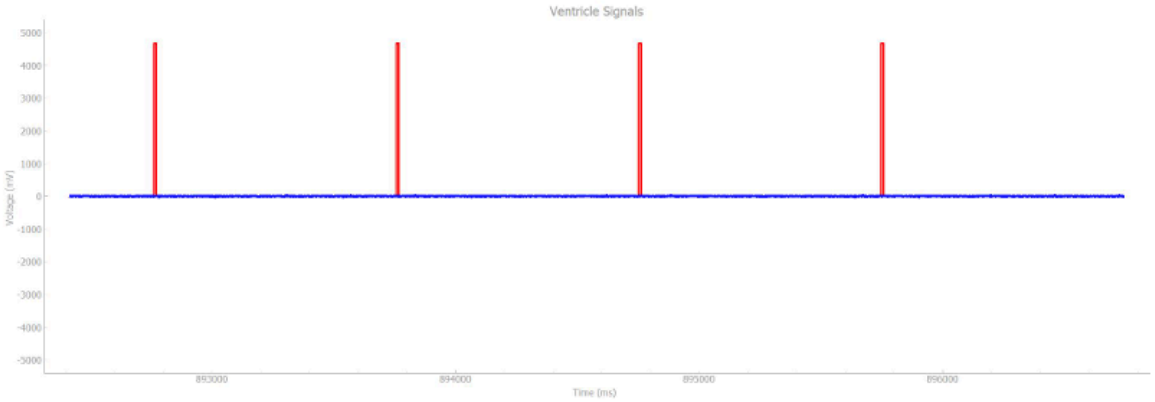
Atrium PW: 0.0 ms

Ventricular PW: 10.0 ms

Heart Rate: 60 BPM

AV Delay: 30 ms

Ventricle Plot:



2.4.2.5 Future Changes

In the future, we may need to sense both the ventricle and atrium at the same time to be able to pace both. This would require us to revamp the sensing architecture to allow foolproof sensing of both chambers.

2.5 Serial Communication

2.5.1 Requirements

The pacemaker serial communication system must allow the user to transmit and receive parameters, ensure reliability, and allow error-free communication using UART at baudrate 115200. Serial communication should allow the pacemaker to modify parameters and modes from a moment to moment basis, not reliant on restarting the device.

2.5.2 Default Parameters

Parameter	Value	Unit
Mode	AOO (1)	N/A
Atrial Pulse Width	1	ms
Ventricle Pulse Width	1	ms
Atrial Amplitude	4	V
Ventricule Amplitude	4	V
Atrial Sensitivity	3	mV
Ventricule Sensitivity	3	mV
ARP	25	ms
VRP	32	ms
Upper Rate Limit	120	ppm
Lower Rate Limit	60	ppm
Response Factor	8	N/A
Reaction Time	30	sec
Recovery Time	5	min
Activity Threshold	90	N/A
MSR	120	ppm

Default parameters are selected based on default values from the Boston Scientific PACEMAKER System Specification document provided to use. Exceptions to this include the Mode parameter which in accordance with the PACEMAKER System Specification document

should be set to DDD, however DDD is not a mode that exists within our pacemaker. Other parameters including ARP and VRP are modified post input such that they can be sent as a unit8.

2.5.2 Serial In and Out

2.5.2.1 Serial In Variables

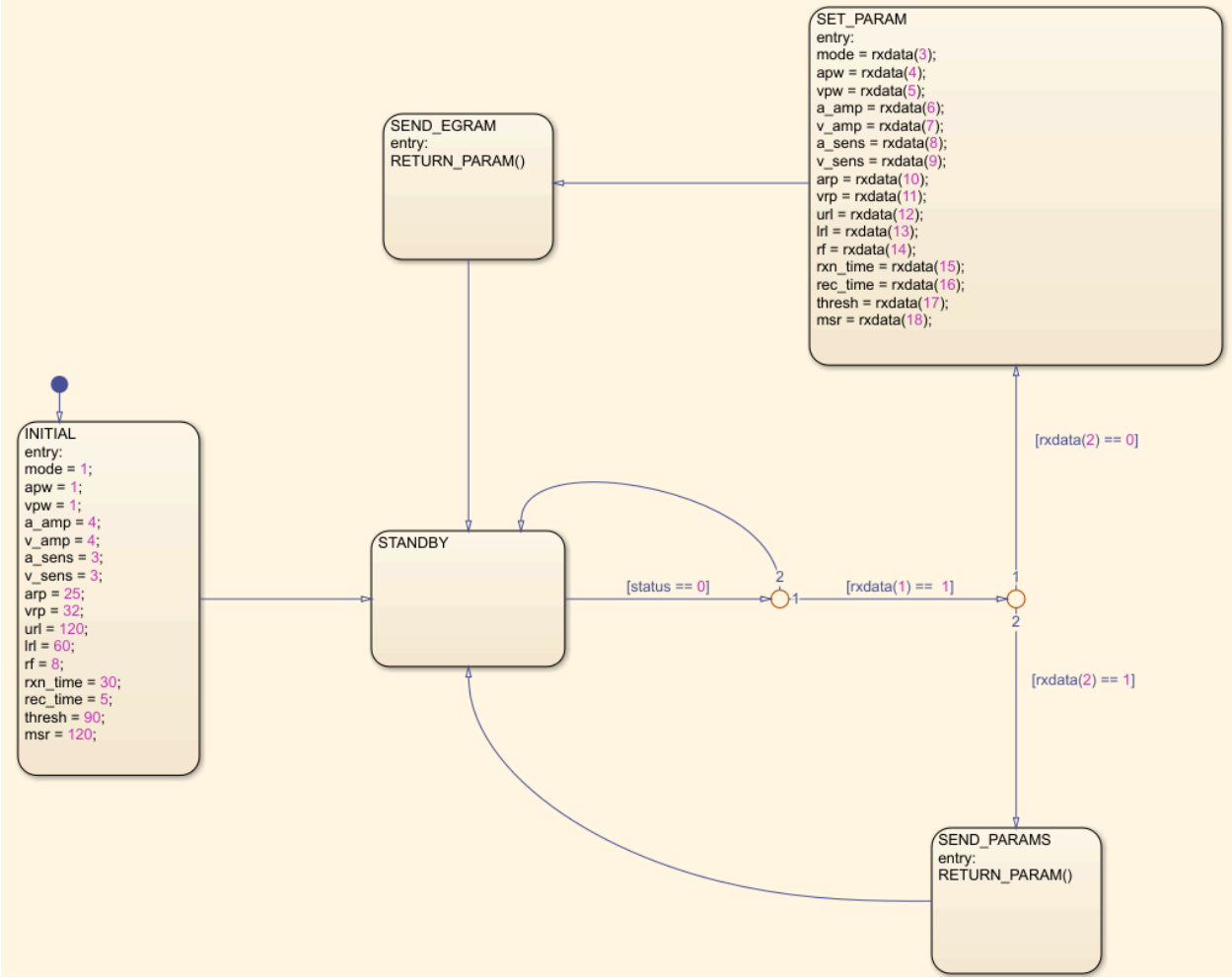
Array Location	Parameter	Data Type	Number of Bytes
rxdata(3)	Mode	uint8	1
rxdata(4)	Atrial Pulse Width	uint8	1
rxdata(5)	Ventricle Pulse Width	uint8	1
rxdata(6)	Atrial Amplitude	uint8	1
rxdata(7)	Ventricule Amplitude	uint8	1
rxdata(8)	Atrial Sensitivity	uint8	1
rxdata(9)	Ventricule Sensitivity	uint8	1
rxdata(10)	ARP	uint8	1
rxdata(11)	VRP	uint8	1
rxdata(12)	Upper Rate Limit	uint8	1
rxdata(13)	Lower Rate Limit	uint8	1
rxdata(14)	Response Factor	uint8	1
rxdata(15)	Reaction Time	uint8	1
rxdata(16)	Recovery Time	uint8	1
rxdata(17)	Activity Threshold	uint8	1
rxdata(18)	MSR	uint8	1

Note: Variables in the above table are identical between input and output, however when outputted they are no longer bound to an array.

Inputs	Definition
Status	Status determines when to communicate with the DCM. Like

	rxdata it is an output from the Serial Receive block.
rxdata(1)	First byte in the array. Is used to check if information has been successfully received. By default rxdata(1) = 0. When it equals zero it cannot return information, when rxdata(1) = 1 data can be returned to the DCM.
rxdata(2)	Second byte in the array. Is used to determine if parameters need to be updated and/or data needs to be returned to the DCM.

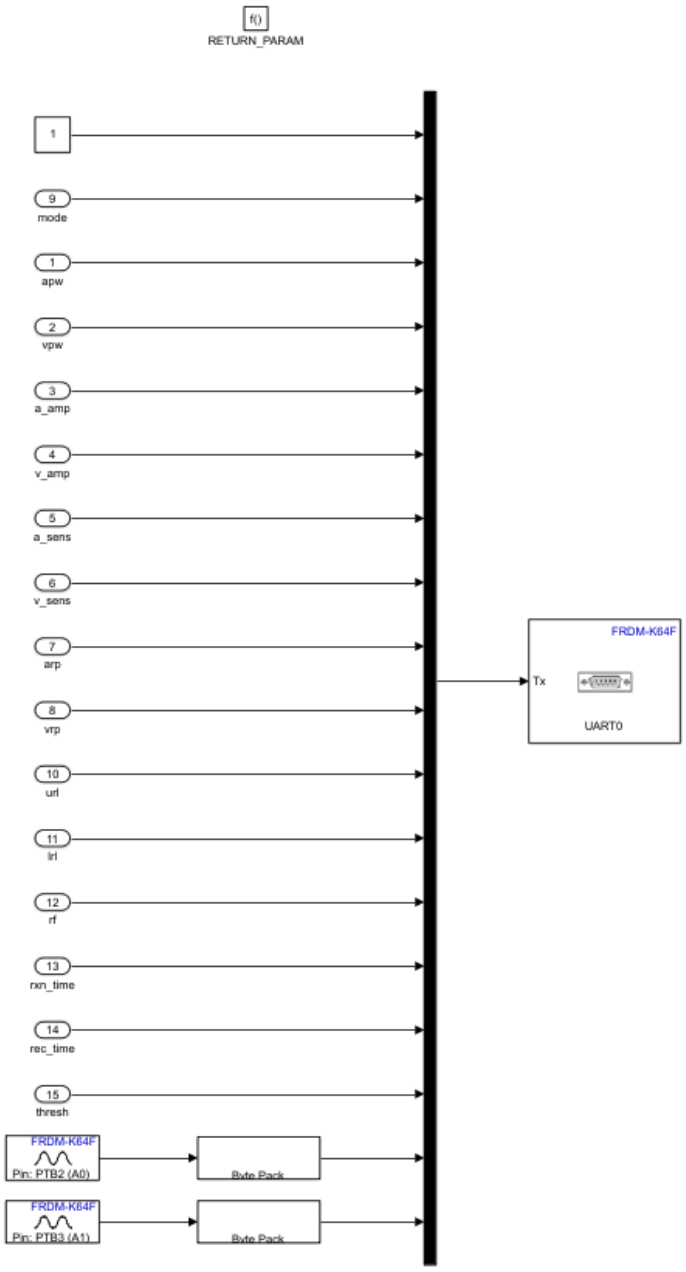
Note: Variables in the above table are only used as an input, they do not have an output.



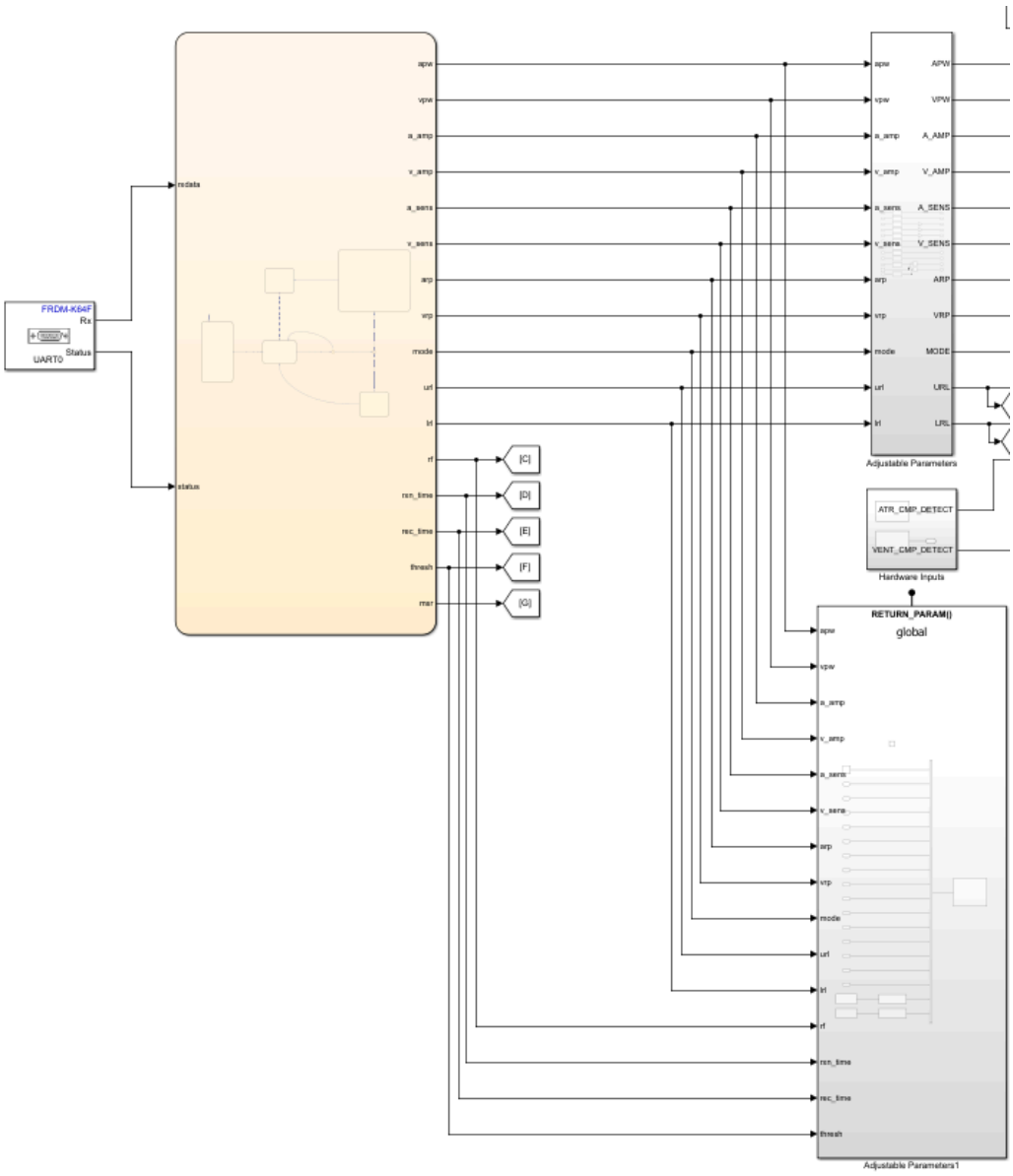
2.5.2.2 Serial Out Variables

Package Array Location	Parameters	Data Type	Number of Bytes
1	Test	uint8	1
2	Mode	uint8	1
3	Atrial Pulse Width	uint8	1
4	Ventricle Pulse Width	uint8	1
5	Atrial Amplitude	uint8	1
6	Ventricle Amplitude	uint8	1
7	Atrial Sensitivity	uint8	1
8	Ventricle Sensitivity	uint8	1
9	ARP	uint8	1
10	VRP	uint8	1
11	Upper Rate Limit	uint8	1
12	Lower Rate Limit	uint8	1
13	Response Factor	uint8	1
14	Reaction Time	uint8	1
15	Recovery Time	uint8	1
16	Activity Threshold	uint8	1
17-24	Atrial Egram Data	double	8
25-32	Ventricle Egram Data	double	8

Note: Variables in the above table are identical between input and output, however when outputted variables become bound to an array.

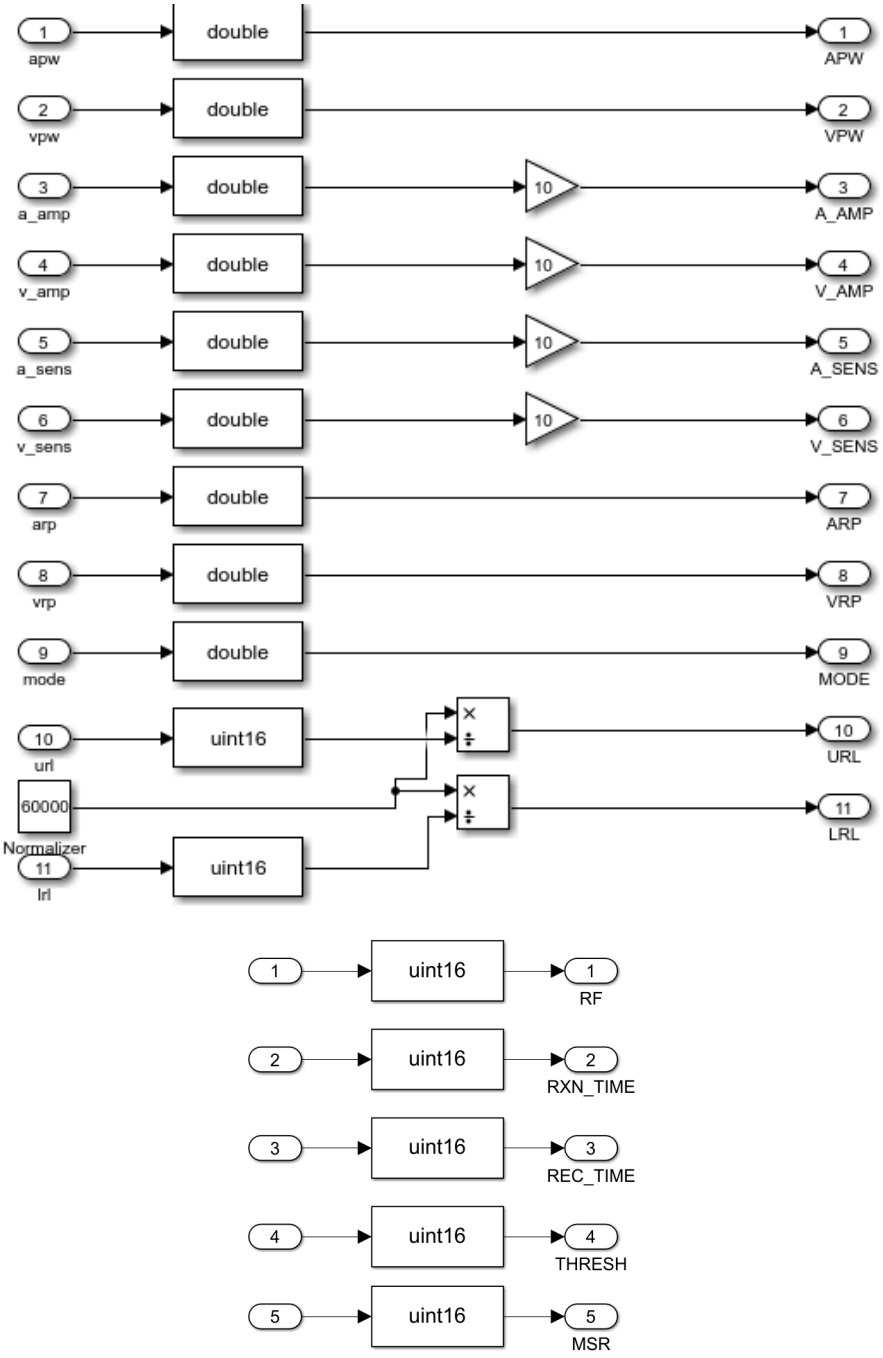

f()
RETURN_PARAM
1
9
mode
1
apw
2
vpw
3
a_amp
4
v_amp
5
a_sens
6
v_sens
7
arp
8
vrp
10
url
11
lrl
12
rf
13
rxn_time
14
rec_time
15
thresh
FRDM-K64F
Pin: PTB2 (A0)
Byte Pack
FRDM-K64F
Pin: PTB3 (A1)
Byte Pack
FRDM-K64F
Tx
UART0

2.5.2.3 Serial Overview



2.5.2.4 Justification of Data Types

Data types were handled after being inputted and outputted as such all data, excluding the atrial and ventricular egram data was handled as uint8. This was done by converting all parameters into doubles or uint16s and making modifications after the conversion. See ‘arp’ in the image below for an example of a modification. Only using bytes simplified communication between DCM and Simulink.



2.5.3 Testing

Parameter	Initial Value	DCM Input	Expected Value	Actual Value	Pass/Fail
Test	N/A	1	1	1	Pass

Mode	1 (AOO)	2 (VOO)	2 (VOO)	2 (VOO)	Pass
Atrial Pulse Width	1	2	2	2	Pass
Ventricle Pulse Width	1	2	2	2	Pass
Atrial Amplitude	4	6	6	6	Pass
Ventricle Amplitude	4	6	6	6	Pass
Atrial Sensitivity	3	4	4	4	Pass
Ventricle Sensitivity	3	4	4	4	Pass
ARP	25	27	27	27	Pass
VRP	32	34	34	34	Pass
Upper Rate Limit	120	130	130	130	Pass
Lower Rate Limit	60	50	50	50	Pass
Response Factor	8	9	9	9	Pass
Reaction Time	30	32	32	32	Pass
Recovery Time	5	7	7	7	Pass
Activity Threshold	90	100	100	100	Pass
MSR	120	130	130	130	Pass

2.6 Pacemaker Validation and Verification

2.6.1 Verification

See **2.3.1 AOO, 2.3.2 VOO, 2.3.3 AAI, 2.3.4 VVI, 2.4.2 Pacing, 2.4.3 Charging, and 2.3.5 Rate Adaptive Pacing(2.3.5.1, 2.3.5.2, 2.3.5.3, 2.3.5.4),** for the tables of the tests and the expected outcomes as well as a walkthrough of the states to find the purpose of each of the modules.

2.6.2 Validation

2.6.2.1 AOO

Test Case*		Expected Result	Actual Result	Status
Natural Atrium	**Natural Ventricle**			
OFF	OFF	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass

*The LRL is set to 60P/M
This is the only test case that we need for AOO, because it only has one purpose which is to give a consistent pulse as the set LRL, without sensing any natural pulse.

2.6.2.2 VOO

Test Case*		Expected Result	Actual Result	Status
Natural Atrium	**Natural Ventricle**			
OFF	OFF	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass

*The LRL is set to 60P/M
This is the only test case that we need for VOO, because it only has one purpose which is to give a consistent pulse as the set LRL, without sensing any natural pulse.

2.6.2.3 AAI

Test Case*				Expected Result	Actual Result	Status
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**			

OFF	OFF	N/A	N/A	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass
ON	OFF	30	1	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
ON	OFF	50	1	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
ON	OFF	50	10	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
ON	OFF	60	1	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M

We need these 5 test cases, because the function of AAI involves more steps.

- The first test case is to see what happens when the atrium isn't working at all, and this is a possibility, the AAI mode should maintain the heart rate of 60BPM all on its own
- The second test case is to see when the heart is pacing far below the LRL. In this case the AAI mode should sense when the heart beats itself, wait for a beat and then send a beat when none is sensed in the required interval. The set heart rate being at 30 BPM means that this should pulse once between 2 pulses.
- The third case is where the natural BPM is slightly below the LRLm but in this case because the pulse width of the natural heart beat is still low enough where the gap between the pulses is greater than the LRL, the pacemaker will pulse once per LRL.
- The fourth case is similar to the third except the pulse width of the heart is larger, large enough to make it so that the gap between the heart pusles is enough to not trigger the pacemaker which is exactly what happens.
- The fifth test case is set to the LRL BPM, and this should not trigger the AAI, since the heart is already pacing at the LRL.

2.6.2.4 VVI

Test Case*				**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**			
OFF	OFF	N/A	N/A	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass

OFF	ON	30	1	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Ventricle once per LRL	Pass
OFF	ON	50	1	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Ventricle once per LRL	Pass
OFF	ON	50	10	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	1	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M
We need these 5 test cases, because the function of VVI involves more steps.

- The first test case is to see what happens when the ventricle isn't working at all, and this is a possibility, the VVI mode should maintain the heart rate of 60BPM all on its own
- The second test case is to see when the heart is pacing far below the LRL. In this case the VVI mode should sense when the heart beats itself, wait for a beat and then send a beat when none is sensed in the required interval. The set heart rate being at 30 BPM means that this should pulse once between 2 pulses.
- The third case is where the natural BPM is slightly below the LRLm but in this case because the pulse width of the natural heart beat is still low enough where the gap between the pulses is greater than the LRL, the pacemaker will pulse once per LRL.
- The fourth case is similar to the third except the pulse width of the heart is larger, large enough to make it so that the gap between the heart pusles is enough to not trigger the pacemaker which is exactly what happens.
- The fifth test case is set to the LRL BPM, and this should not trigger the VVI, since the heart is already pacing at the LRL.

2.6.2.5 AOOR

Test Case			**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Pacemaker Shaking**			
OFF	OFF	Not shaking	Pacemaker pulsing Atrium at LRL	Pacemaker pulsing Atrium at LRL	Pass
OFF	OFF	Moderate Shaking	Pacemaker pulsing Atrium at ADAPTIVERATE	Pacemaker pulsing Atrium at ADAPTIVERATE	Pass

OFF	OFF	High Shaking	Pacemaker pulsing Atrium at URL	Pacemaker pulsing Atrium at URL	Pass

*The LRL is set to 60P/M
*The URL is set to 120P/M

The AOOR mode is meant to be very similar in functionality to the AOO mode where it is expected to pace the heart at LRL if no natural pulse is detected but increase the pulsing rate if the absence of natural pulse is accompanied by a lot of movement i.e. high activity levels. The pulsing rate can be increased up to the URL value.

2.6.2.6 VOOR

Test Case			**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Pacemaker Shaking**			
OFF	OFF	Not shaking	Pacemaker pulsing Ventricle at LRL	Pacemaker pulsing Ventricle at LRL	Pass
OFF	OFF	Moderate Shaking	Pacemaker pulsing Ventricle at ADAPTIVERATE	Pacemaker pulsing Ventricle at ADAPTIVERATE	Pass
OFF	OFF	High Shaking	Pacemaker pulsing Ventricle at URL	Pacemaker pulsing Ventricle at URL	Pass

*The LRL is set to 60P/M
*The URL is set to 120P/M

The VOOR mode is meant to be very similar in functionality to the VOO mode where it is expected to pace the heart at LRL if no natural pulse is detected but increase the pulsing rate if the absence of natural pulse is accompanied by a lot of movement i.e. high activity levels. The pulsing rate can be increased up to the URL value.

2.6.2.7 AAIR

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacemaker shaking**			

OFF	OFF	N/A	N/A	No shaking	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass
ON	OFF	30	1	No shaking	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
ON	OFF	50	1	No shaking	Pacemaker Pulsing Atrium once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
ON	OFF	50	10	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
ON	OFF	60	1	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	Moderate shaking	Pacemaker Pulsing Atrium at ADAPTIVERATE	Pacemaker Pulsing Atrium at ADAPTIVERATE	Pass
ON	OFF	30	1	Moderate shaking	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pass
ON	OFF	50	1	Moderate shaking	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pacemaker Pulsing Atrium once per ADAPTIVERATE	Pass
ON	OFF	50	10	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
ON	OFF	60	1	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	High shaking	Pacemaker Pulsing Atrium at URL	Pacemaker Pulsing Atrium at URL	Pass
ON	OFF	30	1	High shaking	Pacemaker Pulsing Atrium once per URL	Pacemaker Pulsing Atrium once per URL	Pass

ON	OFF	50	1	High shaking	Pacemaker Pulsing Atrium once per URL	Pacemaker Pulsing Atrium once per URL	Pass
ON	OFF	50	10	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
ON	OFF	60	1	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M
*The URL is set to 120P/M
*Moderate shaking is defined as any values from the accelerometer that increase the pacing rate above LRL but don't cause it to reach the URL.
*High shaking is defined as any values from the accelerometer that cause the pacing rate to reach the URL.

We need these 3 activity scenarios which amount to a total of 15 test cases, because the function of AAIR involves more steps:

- The first set of 5 test cases are to see what happens when there is no movement at all i.e. the accelerometer values are zero. However, different natural atrium heart beats are possible which are captured in these 5 test cases. It is worth noting that the results of this test set correspond to that of AAI mode.
- The second set of 5 test cases are to see how the pacemaker responds if there is moderate movement. The mode works in a similar fashion to AAI but is able to change the pacing rate in response to the accelerometer input. The pacing rate stays within [LRL, URL].
- The third set of 5 test cases are to see how the pacemaker responds to a high amount of movement which would imply that the user is engaging in heavy exercise. The mode works in similar fashion to AAI but is able to change the pacing rate to the URL in response to the high values from the accelerometer.
- For a detailed explanation of the working of the AAI mode, please refer to **section 2.3.3**

2.6.2.8 VVIR

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacemaker shaking**			
OFF	OFF	N/A	N/A	No shaking	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass

OFF	ON	30	1	No shaking	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Ventricle once per LRL	Pass
OFF	ON	50	1	No shaking	Pacemaker Pulsing Ventricle once per LRL	Pacemaker Pulsing Atrium once per LRL	Pass
OFF	ON	50	10	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	1	No shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	Moderate shaking	Pacemaker Pulsing Ventricle at ADAPTIVERATE	Pacemaker Pulsing Ventricle at ADAPTIVERATE	Pass
OFF	ON	30	1	Moderate shaking	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pass
OFF	ON	50	1	Moderate shaking	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pacemaker Pulsing Ventricle once per ADAPTIVERATE	Pass
OFF	ON	50	10	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	1	Moderate shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	OFF	N/A	N/A	High shaking	Pacemaker Pulsing Ventricle at URL	Pacemaker Pulsing Ventricle at URL	Pass
OFF	ON	30	1	High shaking	Pacemaker Pulsing Ventricle once per URL	Pacemaker Pulsing Ventricle once per URL	Pass

OFF	ON	50	1	High shaking	Pacemaker Pulsing Ventricle once per URL	Pacemaker Pulsing Ventricle once per URL	Pass
OFF	ON	50	10	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	1	High shaking	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M
*The URL is set to 120P/M
*Moderate shaking is defined as any values from the accelerometer that increase the pacing rate above LRL but don't cause it to reach the URL.
*High shaking is defined as any values from the accelerometer that cause the pacing rate to reach the URL.

We need these 3 activity scenarios which amount to a total of 15 test cases, because the function of VVIR involves more steps:

- The first set of 5 test cases are to see what happens when there is no movement at all i.e. the accelerometer values are zero. However, different natural ventricle heart beats are possible which are captured in these 5 test cases. It is worth noting that the results of this test set correspond to that of VVI mode.
- The second set of 5 test cases are to see how the pacemaker responds if there is moderate movement. The mode works in a similar fashion to VVI but is able to change the pacing rate in response to the accelerometer input. The pacing rate stays within [LRL, URL].
- The third set of 5 test cases are to see how the pacemaker responds to a high amount of movement which would imply that the user is engaging in heavy exercise. The mode works in similar fashion to VVI but is able to change the pacing rate to the URL in response to the high values from the accelerometer.
- For a detailed explanation of the working of the VVI mode, please refer to **section 2.3.3**

2.6.2.9 Serial Communication

Parameter	Initial Value	DCM Input	Expected Value	Actual Value	Pass/Fail
Test	N/A	1	1	1	Pass
Mode	1 (AOO)	2 (VOO)	2 (VOO)	2 (VOO)	Pass
Atrial Pulse Width	1	2	2	2	Pass

Ventricle Pulse Width	1	2	2	2	Pass
Atrial Amplitude	4	6	6	6	Pass
Ventricle Amplitude	4	6	6	6	Pass
Atrial Sensitivity	3	4	4	4	Pass
Ventricle Sensitivity	3	4	4	4	Pass
ARP	25	27	27	27	Pass
VRP	32	34	34	34	Pass
Upper Rate Limit	120	130	130	130	Pass
Lower Rate Limit	60	50	50	50	Pass
Response Factor	8	9	9	9	Pass
Reaction Time	30	32	32	32	Pass
Recovery Time	5	7	7	7	Pass
Activity Threshold	90	100	100	100	Pass
MSR	120	130	130	130	Pass

2.6.2.10 Pacing

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacing Mode**			
OFF	OFF	N/A	N/A	AOO	Pacemaker Pulsing Atrium at LRL	Pacemaker Pulsing Atrium at LRL	Pass

OFF	OFF	N/A	N/A	VOO	Pacemaker Pulsing Ventricle at LRL	Pacemaker Pulsing Ventricle at LRL	Pass

*The LRL is set to 60P/M

These test cases demonstrate the functionality of AOO and VOO as they should both pulse at LRL when no natural pulse is detected

2.6.2.11 Charging/Sensing

Test Case*					**Expected Result**	**Actual Result**	**Status**
Natural Atrium	**Natural Ventricle**	**Heart Rate (BPM)**	**Pulse Width**	**Pacing Mode**			
ON	OFF	60	10.0	AAI	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass
OFF	ON	60	10.0	VVI	Pacemaker not Pulsing	Pacemaker not Pulsing	Pass

*The LRL is set to 60P/M

These test cases demonstrate the functionality of AII and VVI as they should not pulse when the BPM is greater than or equal to the LRL.

3. DCM User Guide and Documentation

3.1 Overview

This section of documentation goes over a comprehensive user guide and any relative security measures, design considerations, and appropriate documents relating to the development and implementation of the Device Controller Monitor (DCM) used as part of Assignment 1 and Assignment 2 of the Pacemaker Project for MECHTRON/SWFRENG 3K04. This can serve as an appropriate manual and guide for those authorized to operate the DCM. The document also assumes some prior knowledge of the associated Pacemaker documents and overview of DCM, but no prerequisite learning is needed to establish enough of an understanding through just this DCM document.

3.2 Program Breakdown

The DCM is programmed using Python 3.8 (64-bit) but has been tested and confirmed to work with all Python versions 3.8 and onwards.

The DCM uses Tkinter and the CustomTkinter library to render the GUI. The CustomTkinter documentation and library were used to create elements, windows, and widgets seen in the program.

Translation from CustomTkinter to Python is handled with specific Tkinter commands and the Tk interface, as it is included with standard Linux, Microsoft Windows, and macOS installs of Python. The name Tkinter comes from the Tk interface.

For any serial communication between the Python program and Simulink program required to document, display, and save patient parameters, PySerial was implemented in these specific areas, as can be seen in the program.

3.3 User Interface (UI)

User interface (UI) is the fundamental intersection of a device's human-computer interaction and communication. Display screens, keyboards, mice, and desktop appearances can all be examples of this. It also refers to how a user uses visual and aural components, like typefaces, buttons, animations, and noises, to engage with an application or website.

There are many different UI toolkits that bind and intertwine with various different coding languages. Since the DCM design is made using python, tkinter is the choice of UI for this application as it is the standard python interface for binding with the GUI Tk toolkit.

3.4 High-Level Program Description

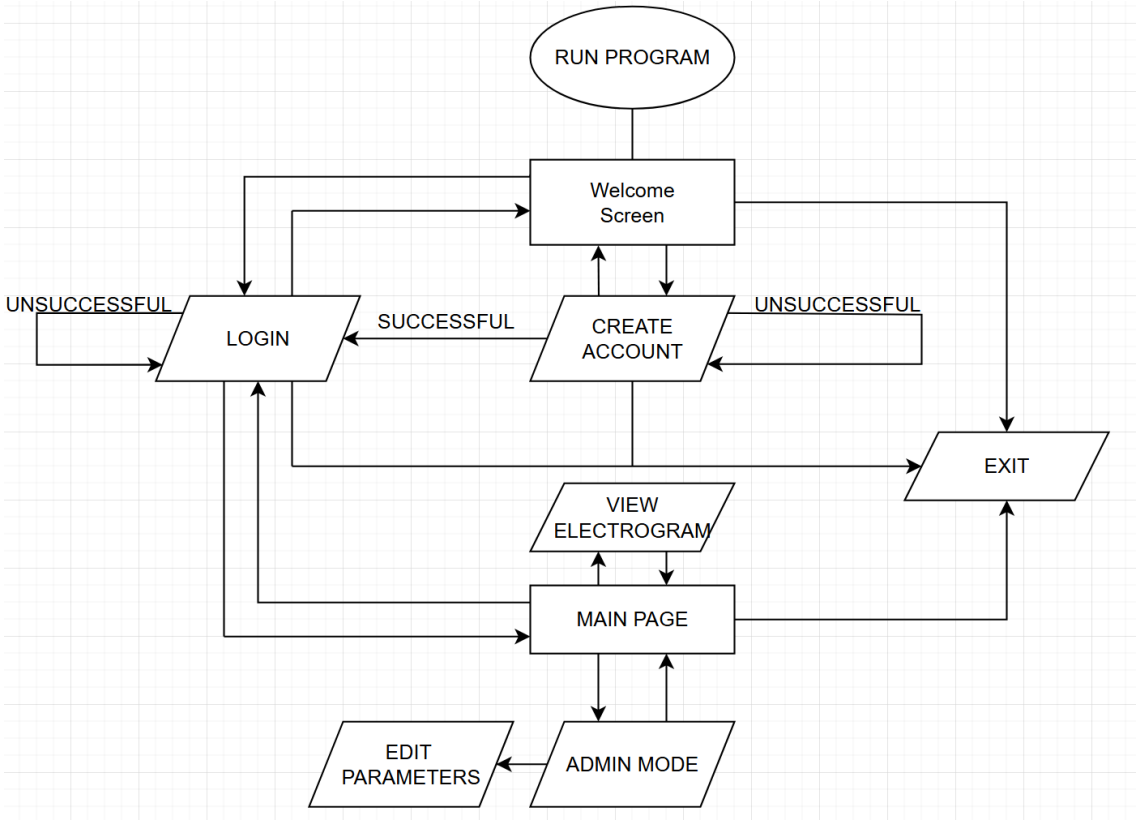
This code is structured to serve as a user interface (UI) for the Device Controller Monitor (DCM) component of a pacemaker project. Its primary purpose is to manage user interactions and provide data encryption, along with a range of tools for monitoring and interfacing with the device in a secure and user-friendly way. The code is organized around a graphical user interface (GUI) constructed with `customtkinter` and `tkinter`, which allows users to interact with the DCM system visually.

A core component of the code is the `UserManager` class, which is responsible for handling user information securely. It saves user data in a `users` folder, encrypting it with the `cryptography` library's Fernet encryption. Users have their data stored in a secure json file, which updates various parameters based on if anything is changed through the DCM or Heartview. When initiated, the `UserManager` class checks for a pre-existing encryption key in a `secret.key` file. If this key is not found, it generates a new key and securely stores it, which ensures that user data remains encrypted and protected from unauthorized access. This process emphasizes security, making sure that data breaches or unauthorized access are minimized.

Additionally, the code integrates `matplotlib` to generate visual plots within the interface. By embedding plots into the GUI with `FigureCanvasTkAgg`, the program can dynamically display information, potentially representing user or device metrics that may be useful for monitoring the pacemaker's operation. The code also imports `serial.tools.list_ports`, indicating it may communicate with hardware, possibly for interfacing with the pacemaker or other peripheral devices.

To ensure secure session handling, the code's encryption mechanism is essential for both protecting stored data and maintaining data integrity when accessed. The GUI, combined with robust data encryption and graphical data presentation, makes this code a comprehensive tool for managing and monitoring pacemaker functionality securely and efficiently. This design enables authorized users to interact with the DCM system intuitively while ensuring that data remains protected throughout its lifecycle.

3.5 High-Level Program Flowchart



3.6 Format of Code

3.6.1 How it Works

Previously using a single file, the Python/Tkinter program works with specific classes and functions inside of them that blend together to store data, render appropriate displays, and allow users to safely and easily work with the program.

Currently for additional modularity, consistency, and safety, the program works with multiple executable files. Each class has its own designated file for extra organization and easy methods of traversing code. There are also folders containing the user database, serial communication, images, and additional graphing to ensure easy accessibility for patients looking at their data.

3.6.2 Imports

The imports all serve specific purposes in ensuring that added parts of the program that can't be achieved directly in Python run properly. To make code from one module available in another in Python, use the import keyword. Python imports are essential for efficiently organizing your code. Making effective use of imports will increase your productivity by enabling code reuse and maintaining the maintainability of your projects.

```
from multiprocessing import connection  # Import the connection module from the multiprocessing library for handling inter-process communication
from pdb import run  # Import the run function from the pdb (Python Debugger) module to facilitate debugging
from tracemalloc import stop  # Import the stop function from tracemalloc to stop memory tracking
import customtkinter as ctk  # Import the CustomTkinter library, which provides custom themed widgets for Tkinter, as 'ctk'
import tkinter as tk  # Import the standard Tkinter library for creating GUI applications, as 'tk'
import os  # Import the os module for interacting with the operating system (file and directory management)
import random  # Import the random module to generate random numbers
from collections import deque  # Import deque from collections for creating a double-ended queue, useful for managing a sequence of items
from PIL import Image, ImageTk  # Import the Image and ImageTk classes from the Pillow library for image processing and display
from matplotlib.figure import Figure  # Import the Figure class from matplotlib for creating figures for plotting
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg  # Import the FigureCanvasTkAgg class to integrate Matplotlib figures into Tkinter
from datetime import datetime  # Import the datetime class to handle date and time operations
import time  # Import the time module for time-related functions (e.g., sleep)
import serial.tools.list_ports  # Import the list_ports module from serial.tools to list available serial ports
from cryptography.fernet import Fernet  # Import the Fernet class from the cryptography library for secure encryption and decryption, particularly for handling passwords
```

See below for additional assignment 2 imports.

```
import json # Provides functions for working with JSON data
import re # Provides functions for working with regular expressions
import serial # Import the serial library to handle serial communication
import struct # Import the struct library to handle binary data packing and unpacking
```

Import Name	Import Purpose
`from multiprocessing import connection`	Handling inter-process communication (primarily for Assignment 2)
`from pdb import run`	Used for debugging (not seen in user code)
`from tracemalloc import stop`	For stopping memory tracking (primarily for Assignment 2)
`import customtkinter as ctk`	CustomTkinter library that allows custom-themed widgets as ctk
`import tkinter as tk`	Standard Tkinter library for GUI applications
`import os`	For operating system interactions like files and directories
`import random`	For generating random numbers
`from collections import deque`	Used to make double-ended queue, important for handling a sequence of items
`from PIL import Image, ImageTk`	For image processing and display from the Pillow library
`from matplotlib.figure import Figure`	Used to create figures for plotting data
`from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg`	For integrating matplotlib features into tkinter; charts of data from the pacemaker
`from datetime import datetime`	For date and time operations such as displaying a specific time
`import time`	For time related functions such as sleep
`import serial.tools.list_ports`	For listing available serial ports on a specific device
`from cryptography.fernet import Fernet`	Using the Fernet class to import cryptography library to disguise and protect passwords
`import json`	Allows functionality with JSON files, which are used in the user database
`import re`	Allows functionality with regular expressions for use in the user pages

`import serial`	The serial library for allowing serial communication with PySerial
`import struct`	The struct library for binary data packing and unpacking

3.6.3 Classes and Functions

In this section the document outlines the formatting and specifications of how the program is divided into its specific classes and functions, with details on how each performs, operates, and has value to the design and specifications of the project.

```
USER_DATA_FILE = "users.txt" # Reference file for handling user data

class UserManager:
    def __init__(self, file_path):
    def _get_or_generate_key(self):
    def _encrypt_password(self, password):
    def _decrypt_password(self, encrypted_password):
    def _sanitize_username(self, username): *
    def _get_user_file_path(self, username): *
    def user_exists(self, username): *
    def read_user(self, username): *
    def save_user(self, username, password): *
    def update_user_data(self, username, data): *
    def delete_user(self, username): *
    def read_all_users(self): *

class LoginPage:
    def __init__(self, master, user_manager, app, success_message=False):
    def create_top_widgets(self): *
    def update_time(self): *
    def create_widgets(self):
    def handle_login(self):
    def login(self, username, password):
    def open_create_user_page(self):

class CreateUserPage:
    def __init__(self, master, user_manager, app):
    def create_top_widgets(self): *
    def update_time(self): *
    def create_widgets(self):
    def handle_create_user(self):
    def show_error(self, message):

class MainPage:
    def __init__(self, master, app, username, user_manager):
    def create_top_widgets(self): *
```

```
    def update_time(self): *
    def create_widgets(self):
    def toggle_admin_mode(self):
    def check_admin_password(self): *
    def delete_current_user_check(self) *
    def send_packet(self, sync, command, mode, apw, vpw, a_amp, v_amp, a_sens,
v_sens, arp, vrp, url, lrl, res_factor, rxn_time, rec_time, thresh): *
    def update_legend(self, labels): *
    def stop_plot(self): *
    def choose_plotting_mode(self): *
    def update_plot_atrial(self, mode): *
    def update_plot_ventrical(self, mode): *
    def update_plot_both(self, mode): *
    def save_graph(self): *
    def show_edit_frame(self):
    def update_edit_frame(self, mode):
    def update_label_and_print(self, label, label_text, slider):
def update_user_data_check(self):
def update_user_data(self):


class App:
    def __init__(self, root):
    def open_login_page(self, success_message=False):
    def open_create_user_page(self):
    def open_main_page(self, username):
    def clear_page(self):

if __name__ == "__main__":
    root = tk.Tk()  # Create the main Tkinter root window
    app = App(root)  # Instantiate the App class
    root.attributes('-fullscreen', True)  # Set the window to fullscreen mode
    # Bind the Escape key to exit full screen mode
    root.bind("<Escape>", lambda event: root.attributes('-fullscreen', False))
    # Start the Tkinter event loop, which keeps the app running and responsive
    root.mainloop()

if __name__ == "__main__":
    root = tk.Tk()
    app = App(root)
    ctk.set_appearance_mode("dark")
```

```
    ctk.set_default_color_theme("dark-blue")
    root.mainloop()
```
*

*: added with Assignment 2

Class Name	Class Purpose
`class UserManager:`	User management operations i.e. tasks done by the user
`class LoginPage:`	For displaying and operating the login page
`class CreateUserPage:`	For showing the create user page and its operations
`class MainPage:`	For displaying and operating the main page
`class App:`	Main controller for GUI as it managers transitions between pages and keeping the flow of user engagements with the program
`if __name__ == "__main__":`	Serves as main function and runs the entire program when executed by the user

Function Name	Function Description
`def __init__(self, file_path):`	Initialize class and its specific traits
`def _get_or_generate_key(self):`	Get or generate a key from a specific file for encryption of passwords
`def _encrypt_password(self, password):`	Encrypt password and store it in users text file for protection
`def _decrypt_password(self, encrypted_password):`	Use the cypher to decrypt the password when it is needed to read for login
`*def read_users(self):`	Read the users file for storing the usernames and encrypted passwords
`*def save_user(self, username, password):`	Save a user by encrypting the password, opening the users file and writing the new username and password to it
`def _sanitize_username(self, username):`	Sanitize the username of the user to create a safe filename to be stored securely in the user json database
`def _get_user_file_path(self, username):`	Get the file path for the user's json file

`def user_exists(self, username):`	Check if the user exists
`def read_user(self, username):`	Read a user's data from their designated json file
`def save_user(self, username, password):`	Save a new user to their own json file in the user database
`def update_user_data(self, username, data):`	Update a user's specific data in their json file in the database
`def delete_user(self, username):`	Delete a user's json file
`def read_all_users(self):`	Read all of the user's data

Function Name	Function Description
`def __init__(self, master, user_manager, app, success_message=False):`	Initialize class and its specific traits
`def create_widgets(self):`	Creates widgets specific to the login page (includes window, buttons, labels, etc.)
`def handle_login(self):`	Get the username and password from the entry boxes entered by the user and check if they are correct
`def login(self, username, password):`	Read the list of users from the file and determine if login can be successful or not
`def open_create_user_page(self):`	Open the page to create a new user if the prompt is pressed
`def create_top_widgets(self):`	Create the widgets that exists at the top bar of the page
`def update_time(self):`	Update the time displayed on the page

Function Name	Function Description
`def __init__(self, master,`	Initialize class and its specific traits

`user_manager, app):`	
`def create_widgets(self):`	Creates widgets specific to the create user page (includes window, buttons, labels, etc.)
`def handle_create_user(self):`	Gets the new username and password and checks if it meets requirements to successfully create a new user
`def show_error(self, message):`	Raise error message if creating user is unsuccessful
`def create_top_widgets(self):`	Create the widgets that exists at the top bar of the page
`def update_time(self):`	Update the time displayed on the page

Function Name	**Function Description**
`def __init__(self, master, app, username, user_manager):`	Initialize class and its specific traits
`*def check_microcontroller(self):`	Continuously checks for connected serial ports from a microcontroller
`def create_widgets(self):`	Creates widgets specific to the main page (includes window, buttons, labels, etc.
`def toggle_admin_mode(self):`	Change the admin mode to on/off based on if user enters correct pin to access it
`*def delete_current_user(self):`	Delete the current logged in user (username and password) from the text file and return to the login page
`*def reset_plot(self):`	Reset the plot being used by the electrogram
`*def segment_button_callback(self, value):`	Switch between the Show Electrogram and Edit Parameters tab based on which button is pressed down
`*def show_electrogram(self):`	Display the electrogram frame with the plot if it is currently toggled
`def show_edit_frame(self):`	Show the edit frame in the specified grid if it is currently toggled

`def update_edit_frame(self, mode):`	Update the edit frame parameters based on which mode from the drop down is pressed
`def update_label_and_print(self, label, label_text, slider):`	Update the label and text from the edit frame slider based on what parameters and values are changed
`def update_plot(self):`	Update the label text with the slider's current value
`def create_top_widgets(self):`	Create the widgets that exists at the top bar of the page
`def update_time(self):`	Update the time displayed on the page
`def check_admin_password(self):`	Determine if the admin mode password was entered correctly
`def delete_current_user_check(self)`	Determine if the user wants to confirm that they want to delete their user
`def send_packet(self, sync, command, mode, apw, vpw, a_amp, v_amp, a_sens, v_sens, arp, vrp, url, lrl, res_factor, rxn_time, rec_time, thresh):`	
`def update_legend(self, labels):`	Dynamically update the legend
`def stop_plot(self):`	Stop the plot from running
`def choose_plotting_mode(self):`	Choose plotting mode for electrogram graph
`def update_plot_atrial(self, mode):`	Update the atrial plot
`def update_plot_ventrical(self, mode):`	Update the ventricle plot
`def update_plot_both(self, mode):`	Update both the atrial and ventricle plot
`def save_graph(self):`	Save the graph to a folder of the user's choosing
`def update_user_data_check(self):`	Confirm that the user wants to update the data from their database
`def update_user_data(self):`	Update the user's data with the changes made

Function Name	Function Description
`def __init__(self, root):`	Initialize class and its specific traits
`def open_login_page(self, success_message=False):`	Opens the login page
`def open_create_user_page(self):`	Opens the create user page
`def open_main_page(self, username):`	Opens the main page
`def clear_page(self):`	Clears the old page by destroying all widgets to ensure a fresh new page is loaded

*: functions used in Assignment 1 that are not used in Assignment 2.

3.7 DCM Requirements

Listed below are the functionality requirements for assignment 1. These requirements encompass outlined essential features for an operable DCM as well as additional features agreed upon by the group.

3.7.1 Login Screen

The login screen must feature options to exit the program, log into the program, and navigate to the create new user page.

The user can log in after successfully filling in the prompted username and password with an existing account. If either is incorrect, the login will fail, and notify the user of the error. The password will be displayed as asterisks when entered for additional security.

3.7.2 Create New User Screen

The create new user screen allows the user to return to the login screen, exit the program, and create a new account.

New accounts are created by inputting a valid username and password. For a username and password to be valid they cannot include any special characters or have an identical username to an existing account.

Passwords displayed on the page will appear as asterisks for security purposes. When a password is stored, it is encrypted.

Invalid attempts to create a new account will prompt the user with an error message. Valid attempts to create a new account will prompt the user of the success and automatically return the user to the login page.

The database of the program can store a maximum of 10 users

3.7.3 Main Screen

The main screen should allow the user to switch between modes AOO, VOO, AAI, and VVI; preview and edit parameters; view the electrogram; export data; send information to the pacemaker; log out; delete a user; exit the program; and toggle Admin Mode on and off

Admin Mode serves as an additional security measure, restricting certain features when turned off. When Admin Mode is off, users are unable to send information to the pacemaker or adjust parameters.

The adjustable parameters include LRL, URL, Atrial Amplitude, Ventricular Amplitude, Atrial Pulse Width, Ventricular Pulse Width, Atrial Sensitivity, Ventricular Sensitivity, ARP, VRP, Hysteresis, and Rate Smoothing. The adjustable parameters are dependent on the selected mode. The parameters have a range of valid selectable values.

When parameters are adjusted in Admin Mode, the variables that store this information are updated accordingly.

The electrogram must display a dynamic plot. Additionally, the main page should display account information, the current time, and the connection status of the pacemaker.

3.7.4 Future Design Requirements v1

The DCM has several requirements that will need to be added. These future requirements are necessary for a functional DCM, and functional UI.

The UI must connect to the pacemaker and properly send and receive data to the pacemaker. The DCM should be able effectively to graph real-time electrograms using the data it receives.

The DCM should also be capable of rejecting invalid parameters, such as instances where the Lower Rate Limit exceeds the Upper Rate Limit.

A UI element should be implemented to prompt for the admin password when entering Admin Mode. This will add an important layer of security and ensure that only authorized users can access critical functions. These improvements will collectively create a more efficient and user-friendly DCM.

The variable storing method should be updated such that parameter settings are saved to the correct account, this can be done by saving a list of the parameters to the same file the usernames and passwords are saved to.

3.7.v2 DCM Requirements

The following section outlines the functional and operational requirements for Assignment 2. These requirements define the essential features for an operable Digital Circuit Model (DCM) of the pacemaker, as well as additional features agreed upon by the development team.

3.7.1.v2 Login Screen

The user must be able to view the battery status of the pacemaker at the login screen.

The user must be able to see the number of users currently saved in the system.

The system must ensure the pacemaker is connected before allowing the user to log in.

3.7.2.v2 Create New Users Screen

Data for newly created user accounts must be stored in a JSON file.

Each JSON file must include, personal user information (e.g., name, ID) and details of all available pacing modes and their associated parameters.

3.7.3.v2 Main Page Screen

Only administrative users are allowed to delete existing user accounts or override saved data with new parameter settings. To secure these operations, access to Admin Mode is password-protected, and critical actions such as saving or deleting data require additional user confirmation to prevent accidental execution.

The main page also includes functionality for managing the electrogram (Egram) graph. Users must be able to start and stop graphing activities, save graphs to a file, and choose to display atrial, ventricular, or both signal types.

Furthermore, the system’s mode selection has been expanded to include AOOR, VOOR, AIIR, and VIIR modes. To accommodate the increasing number of editable parameters, the parameter frame has been designed to be scrollable, allowing users to view and edit all settings seamlessly within the same interface.

3.7.4.v2 Pyserial Communication

The data displayed on the Egram graph must accurately represent the signals transmitted from Simulink and must adhere to correct timing to emulate real-time functionality.

User parameters and pacing modes should be reliably sent to the pacemaker, and the system must validate the integrity of transmitted data by comparing sent and received values. In the event of a discrepancy, the system should notify the user of the error to enable corrective action.

The software must also be capable of distinguishing between situations where new data is sent to the pacemaker and instances where it only receives data, ensuring that new data is transmitted only when explicitly triggered by the admin user.

3.7.5.v2 Future DCM Requirements

A password recovery mechanism and two-factor authentication should be implemented to enhance login security.

The system should provide storage for historic user settings and pacing outputs, allowing reference and analysis of past configurations. To prevent accidental data loss, a backup mechanism should be incorporated, enabling recovery of deleted user accounts and settings.

Additional improvements, such as accessibility features and intuitive navigation, may also be considered to ensure the system is inclusive and user-friendly.

3.8 DCM Design Decisions

3.8.1.v1 Windows

The UI is designed around 3 specified states, each having its own window. These states are Login, Create User, and Main Page. Only one state can be active at a time

The default size of the window is full screen. The window can be adjusted from full screen to windowed. When windowed elements adjust to fit the new size.

3.8.2.v1 Variables

Variables are stored in a separate file from the program. This file saves usernames and encrypted passwords. Using a file prevents accounts from being lost upon closing the program.

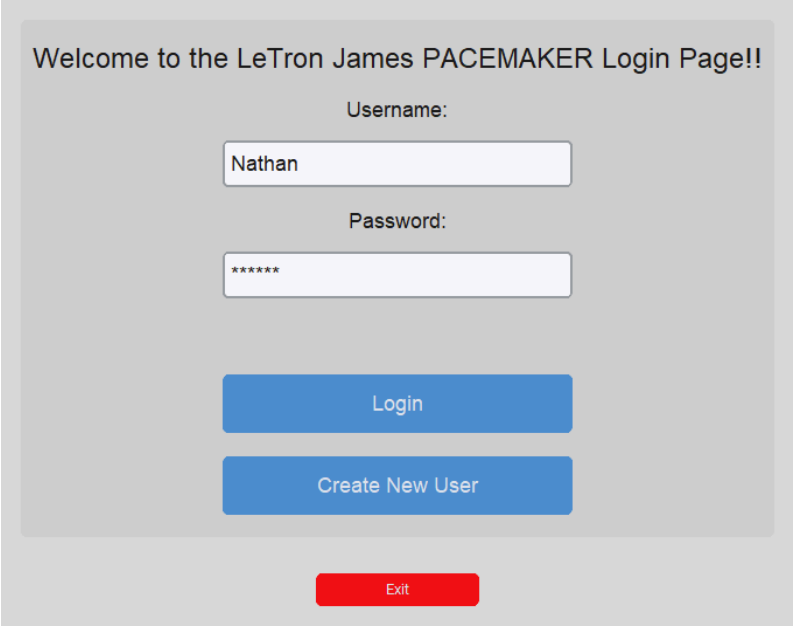
The parameters are currently not linked to specific users and are not saved in a separate file. Restarting the program will cause the parameters to return to their default settings.

3.8.3.v1 Login

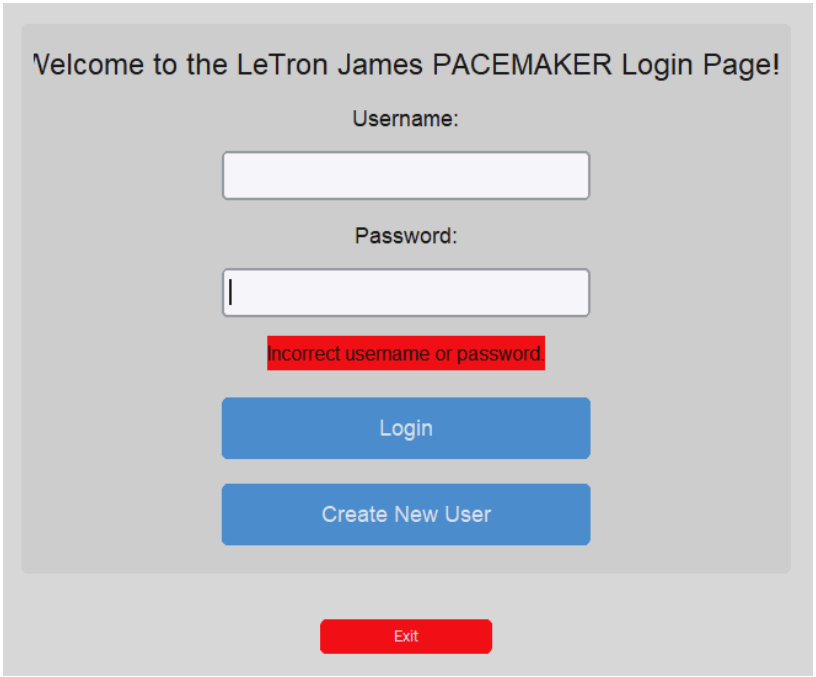
The login interface consists of five interactive elements. Two text boxes for Username and Password, and three buttons; Login, Create New User, and Exit.

The Username and Password text boxes, along with the Login and Create New User buttons, are grouped together as they all relate to accessing the DCM.

In contrast, the Exit button is visually distinct, as functionally it closes the program. This button is colored in a red shade to further emphasize its separation from the other elements, which are primarily blue.

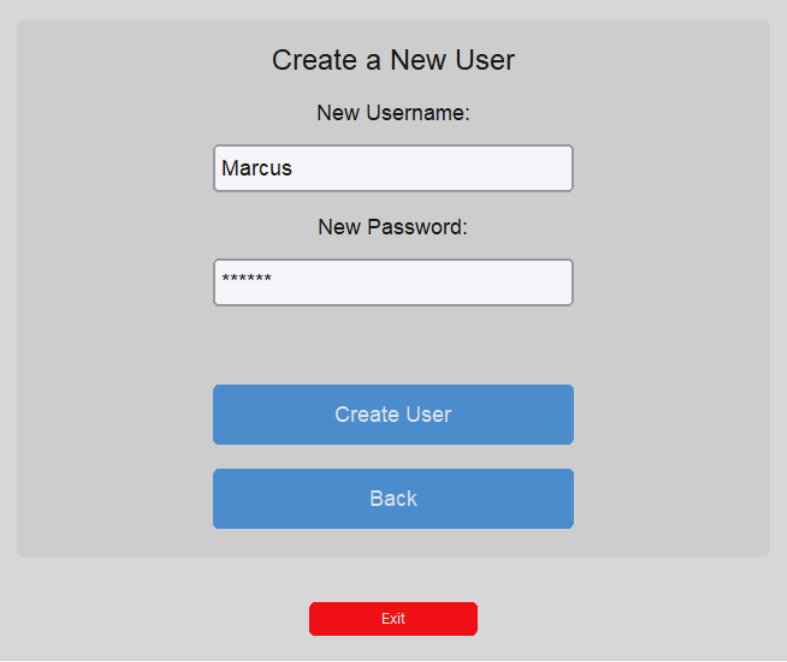


Additionally, non-interactive text is displayed at the top of the window to indicate the name of the program. Color-coded messages also appear based on user input: red text indicates invalid entries for the username and/or password, while green text signifies that a new account has been successfully created.



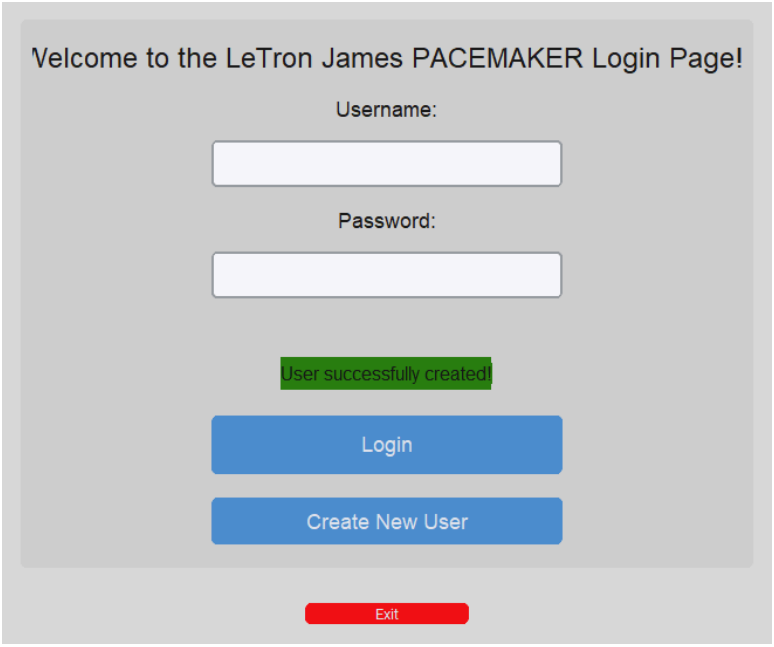
3.8.4.v1 Create New User

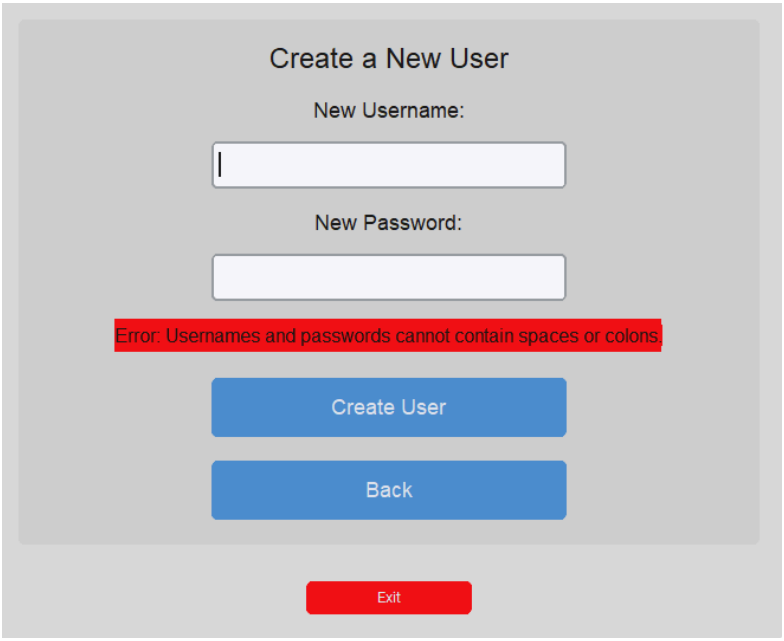
The Create New User interface features a text element at the top of the page which indicates the user is in the Create New User window. Additionally, there are five interactive elements. Two text boxes; New Username, and New Password, and three buttons; Create User, Back, and Exit.



New Username, New Password, Create User, and Back are grouped together, sharing the same functionality, returning you to the Login page.

When the user fills out the New Username and New Password text boxes correctly, the Create User button will successfully return them to the login page and inform the user the account was created successfully. If there are any issues with the inputs, a red prompt will appear to inform the user of the problem. An error will not allow you to return to the Login page with the Create User button.



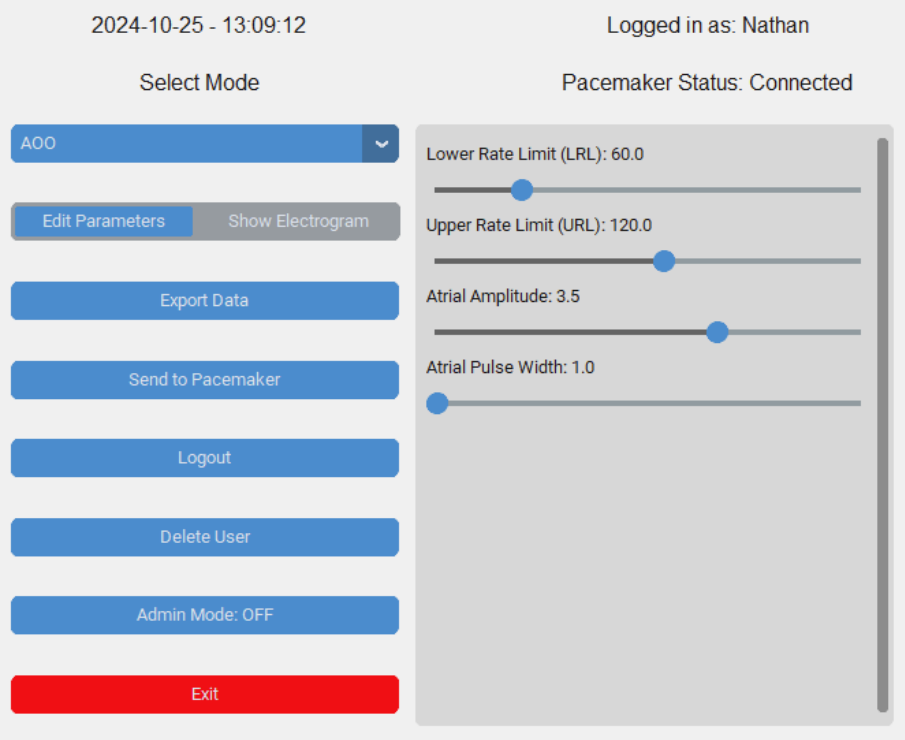


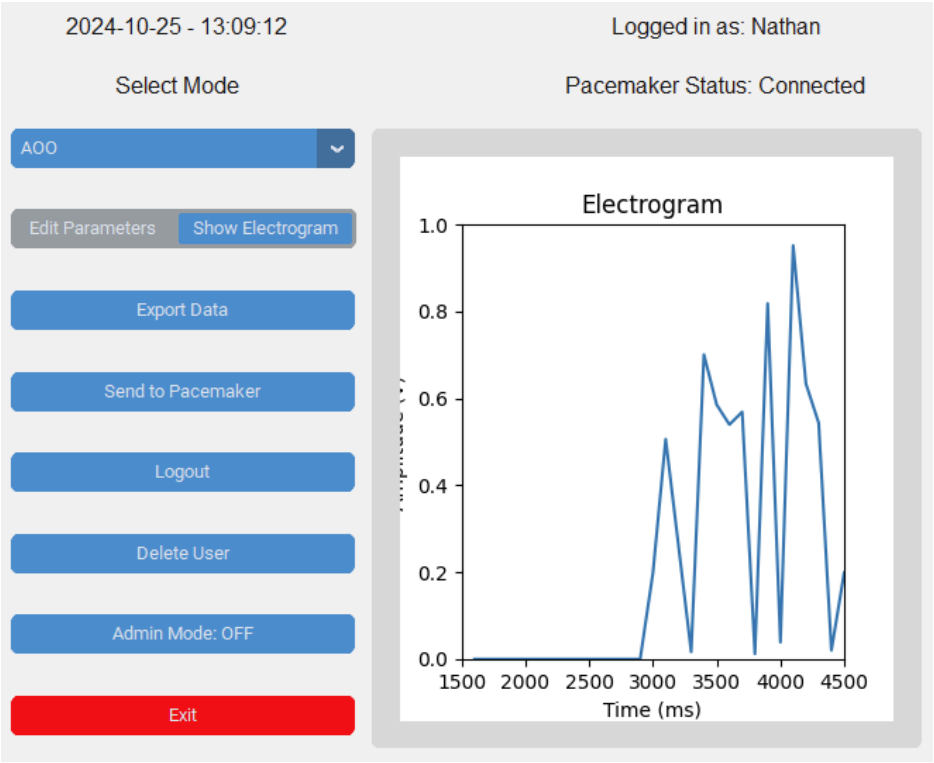
The back button returns the user to the login page but does not create a new account.

The Exit button is disjointed, as it serves the separate function of closing the program. This button is colored in a red shade to further emphasize its separation from the other elements, which are primarily blue.

3.8.5.v1 Main Page

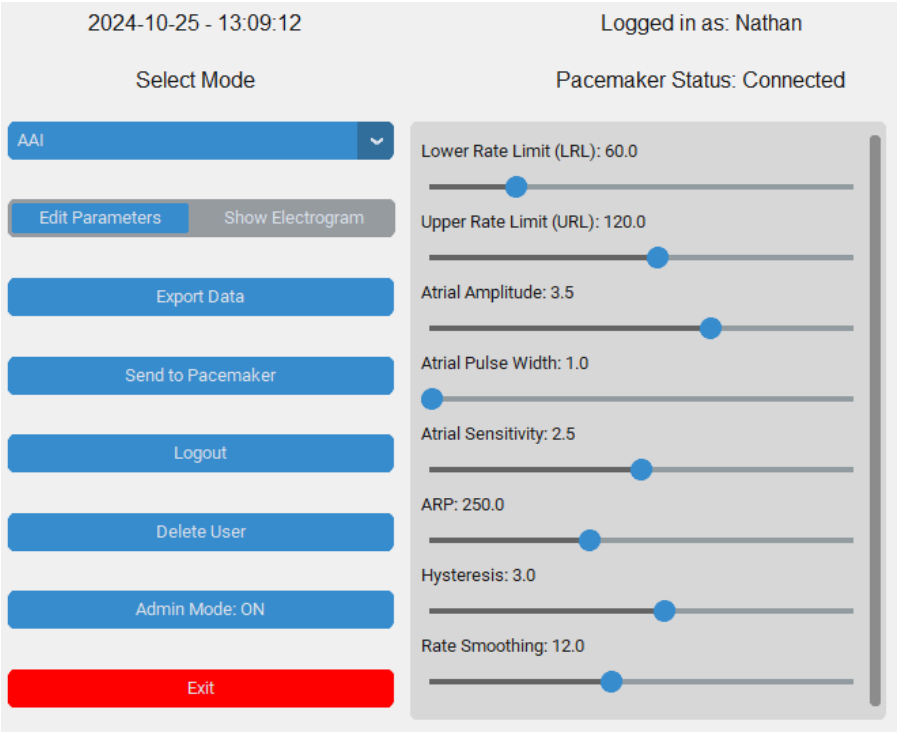
The Main Page is divided into three sections: informative text at the top, interactive buttons on the left, and a parameter editor/electrogram graph on the right.





The top portion displays information, including the time, the current account username, and the connection status of the pacemaker.

The left side features several interactive buttons: a drop-down mode selector for AOO, VOO, AAI, and VVI, and a toggle switch between "Edit Parameters" and "Show Electrogram." Also included are six buttons for exporting data, sending commands to the pacemaker, logging out, deleting a user, entering admin mode, and exiting the program. All buttons are blue with the exception of “Exit” which is a different color, red.



The toggle allows users to switch between adjusting parameters and viewing the electrogram. When "Edit Parameters" is selected, adjustable sliders for each parameter are displayed, with intrinsic max ranges set for each. The specific parameters shown will vary depending on the selected mode. Conversely, when "Show Electrogram" is selected, the parameter sliders will be replaced by a graph that displays real-time data exchanged between the DCM and the pacemaker.

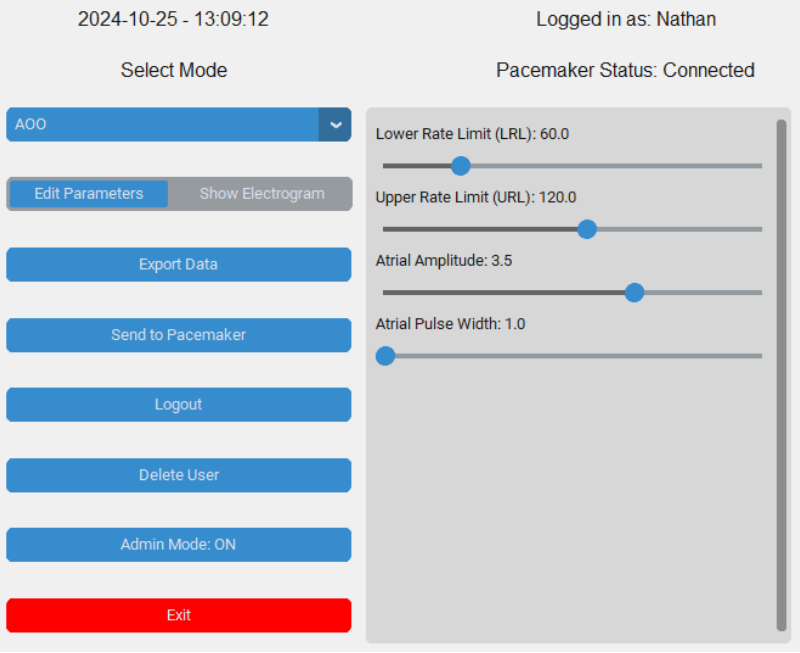
Admin Mode is a toggleable feature that is off by default. When Admin Mode is disabled, users are restricted from deleting accounts, adjusting parameters, sending information to the pacemaker, and exporting data. This mode is designed to be password-protected, preventing accidental changes to parameters and ensuring that only authorized users can access critical functions.

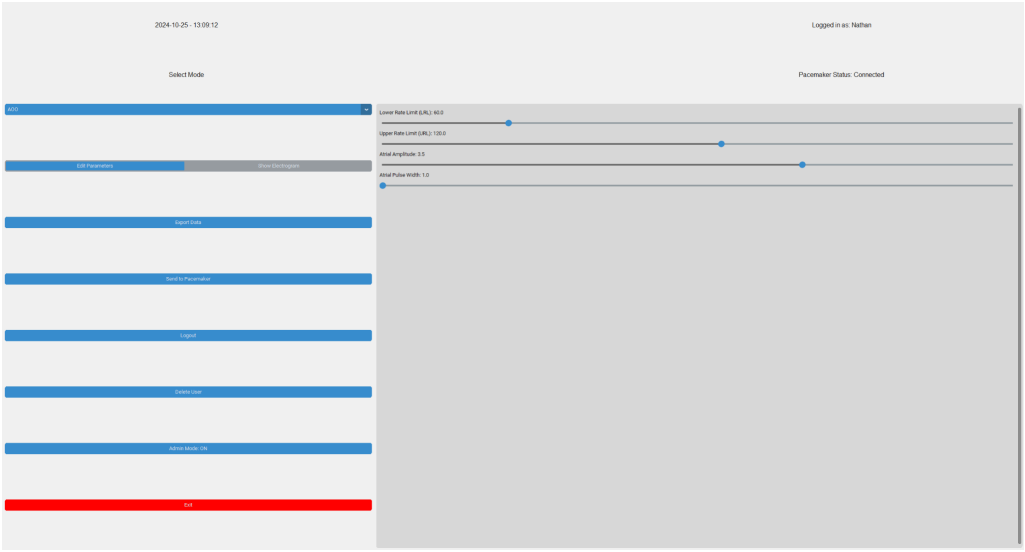
3.8.6.v1 Future DCM Design Decisions

The DCM has several areas that require improvement to enhance user experience and functionality before the full release and presentation of it after Assignment 2.

Both the login and create user pages should display the current number of users. This change is intended to improve quality of life by providing users with relevant account information at a glance.

Additionally, limiting excessive empty space when in fullscreen will make the UI more user-friendly and visually appealing. Currently, when the window is made smaller the buttons fill more space, this should be the case when the window is large too.





Button organization needs to be adjusted to align with industry standards. For example, positioning the Exit button in the top right corner and the time display in the bottom right will meet users' expectations and improve navigation.

Furthermore, grouping and color coding related buttons, notably on the main page, will enhance the overall user experience. The current button layout lacks thoughtful organization and can lead to user errors. Logout, Delete User, and Exit should be grouped together as functionally they serve to leave the main page.

To ensure that parameter changes are made intuitively, the "Send to Pacemaker" button should be linked in the edit parameters frame.

Additionally, buttons accessible only in admin mode should appear grayed when not in admin mode to indicate they are currently non-functional and to avoid confusion.

3.8.7.v2 Implemented DCM Design Decisions

3.8.7.1.v2 Variables

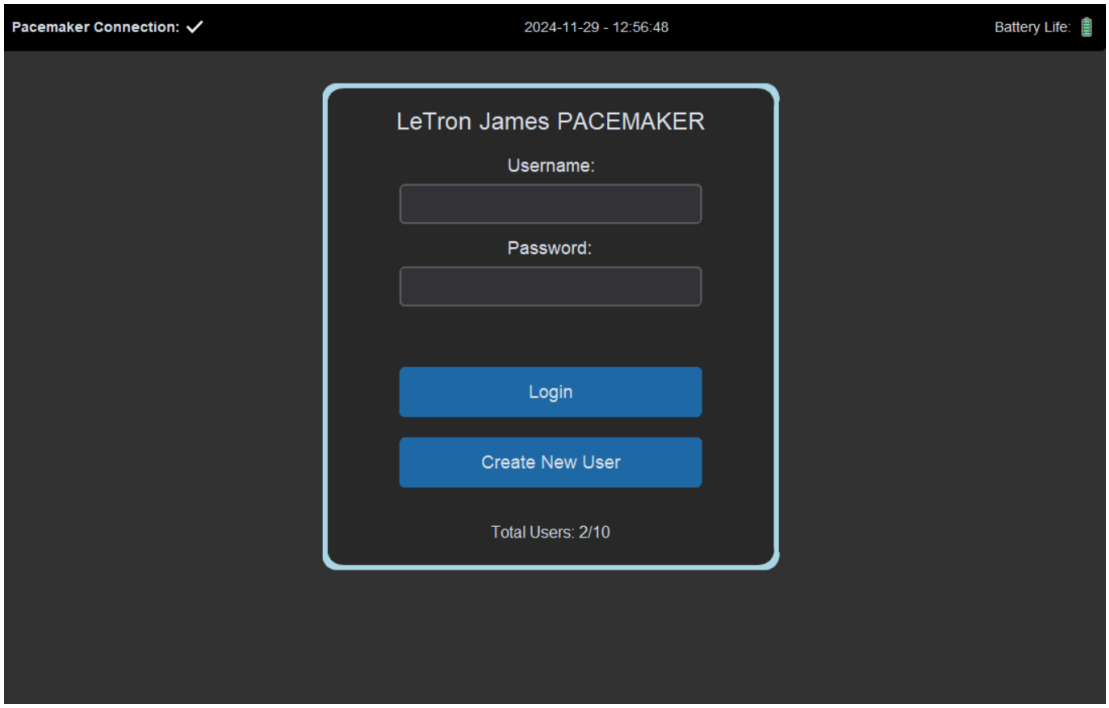
The DCM stores variables in a dedicated folder separate from the main program files. User data is saved as individual JSON files, with each file named after the user's unique username. Passwords within these files are encrypted to enhance security. This system ensures that user accounts remain intact even after the program is closed. By using JSON as the storage format, the data is both secure and easy to interpret, offering a user-friendly yet robust solution. Moreover, storing user accounts in separate files ensures that if one file is corrupted or lost, other accounts remain unaffected, adding an additional layer of resilience to the system.


```
{
    "username": "aidan",
    "password": "gAAAAABnSf-SYXGtuKtr-vRpOO76x-F_Y4TtlHm9n8TbjQre8jwa7uNimVkQMUklvkHT3ZrctO8YvFbfFRmfdVUItvhJajfBVw==",
    "AOO": {
        "Lower Rate Limit": 60,
        "Upper Rate Limit": 120,
        "Atrial Amplitude": 3.5,
        "Atrial Pulse Width": 1
    },
    "VOO": {
        "Lower Rate Limit": 60,
        "Upper Rate Limit": 120,
        "Ventricular Amplitude": 3.5,
        "Ventricular Pulse Width": 1
    },
    "AAI": {
        "Lower Rate Limit": 60,
        "Upper Rate Limit": 120,
        "Atrial Amplitude": 3.5,
        "Atrial Pulse Width": 1,
        "Atrial Sensitivity": 2.5,
        "ARP": 250,
        "Hysteresis": 3.0,
        "Rate Smoothing": 12
    }
```

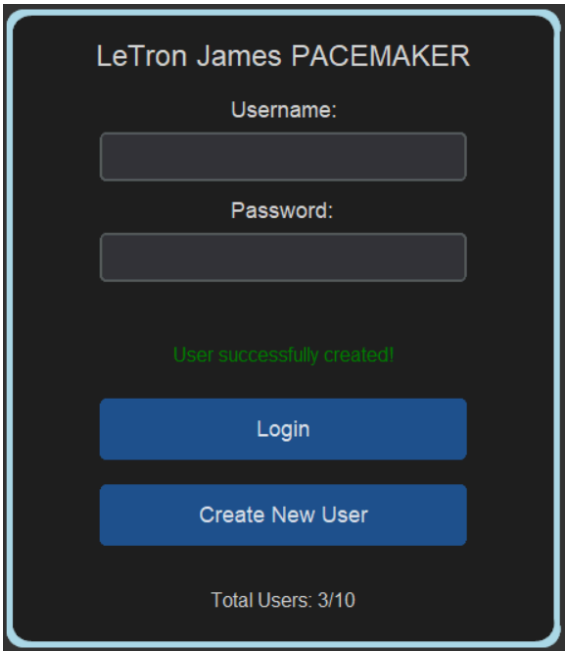

3.8.7.2.v2 Login

The login interface comprises four interactive elements: two text boxes for username and password input, and two buttons for logging in or creating a new user. These elements are grouped together to streamline the process of accessing the DCM. To enhance usability, the current number of saved users is displayed prominently on the login page. If the maximum user capacity is reached, the count is highlighted in red to alert users visually.

A consistent banner at the top of the login page provides general information, including battery status, pacemaker connection status, and the current date and time. Additionally, the program's name is displayed as non-interactive text at the top of the screen.

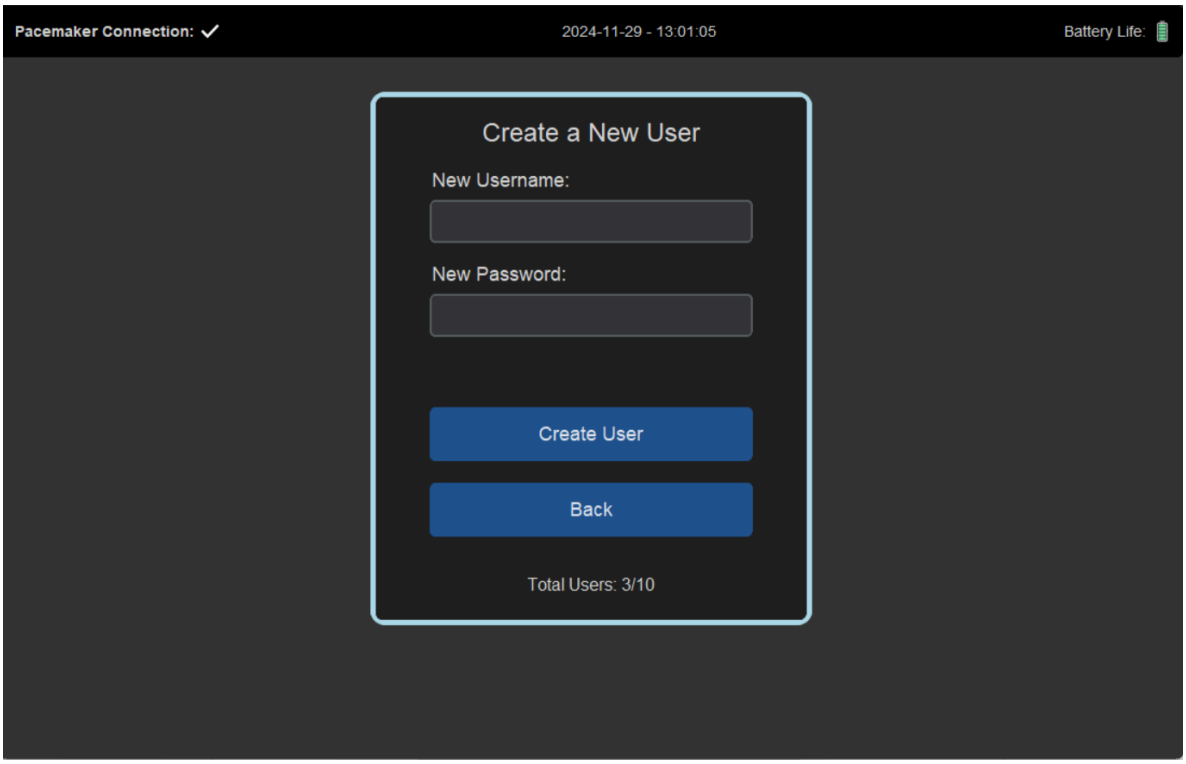


User feedback is implemented through color-coded messages: red text indicates invalid login credentials, while green text confirms successful account creation, ensuring clear and immediate communication with the user.



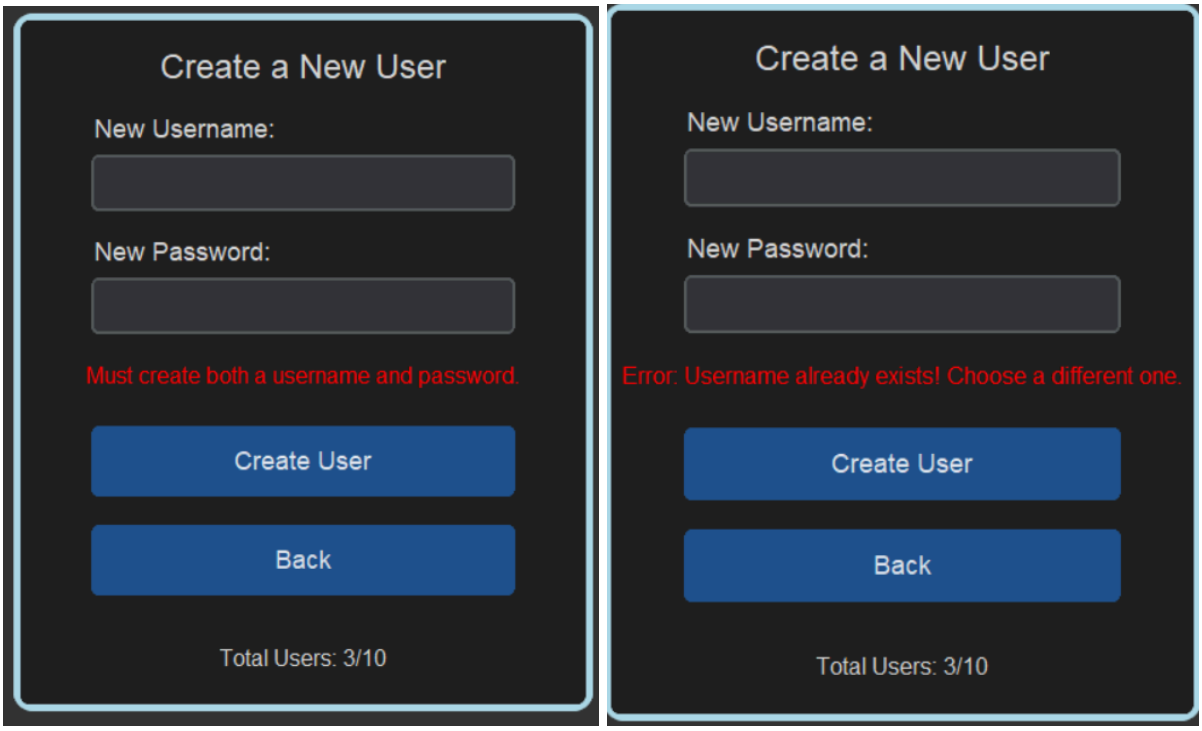
3.8.7.3.v2 Create New User

The Create New User page is designed to guide users seamlessly through the account creation process. At the top of the page, a static text element informs users that they are in the account creation window. This interface includes four interactive elements: text boxes for entering a new username and password, and buttons for creating the account or returning to the login page. The current total number of users is also displayed in this window for reference.



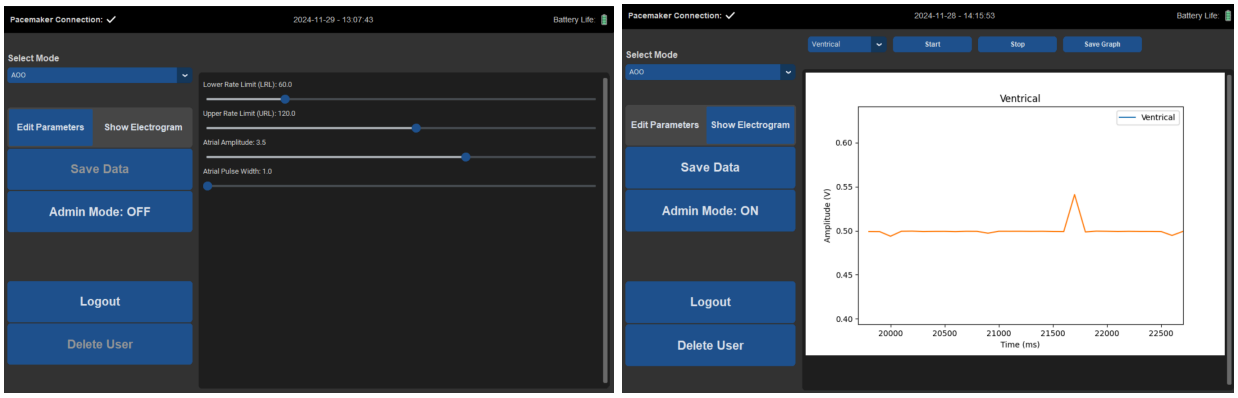
New Username, New Password, Create User, and Back are grouped together, sharing the same functionality, returning you to the Login page.

When users enter valid inputs into the text boxes and click the "Create User" button, the system successfully creates the account, returns the user to the login page, and provides a green confirmation message. If there are issues with the inputs, such as a username already in use or an invalid password, a red prompt notifies the user of the problem and provides guidance on how to resolve it. The "Back" button allows users to return to the login page without creating an account, giving them the flexibility to exit the process if needed.

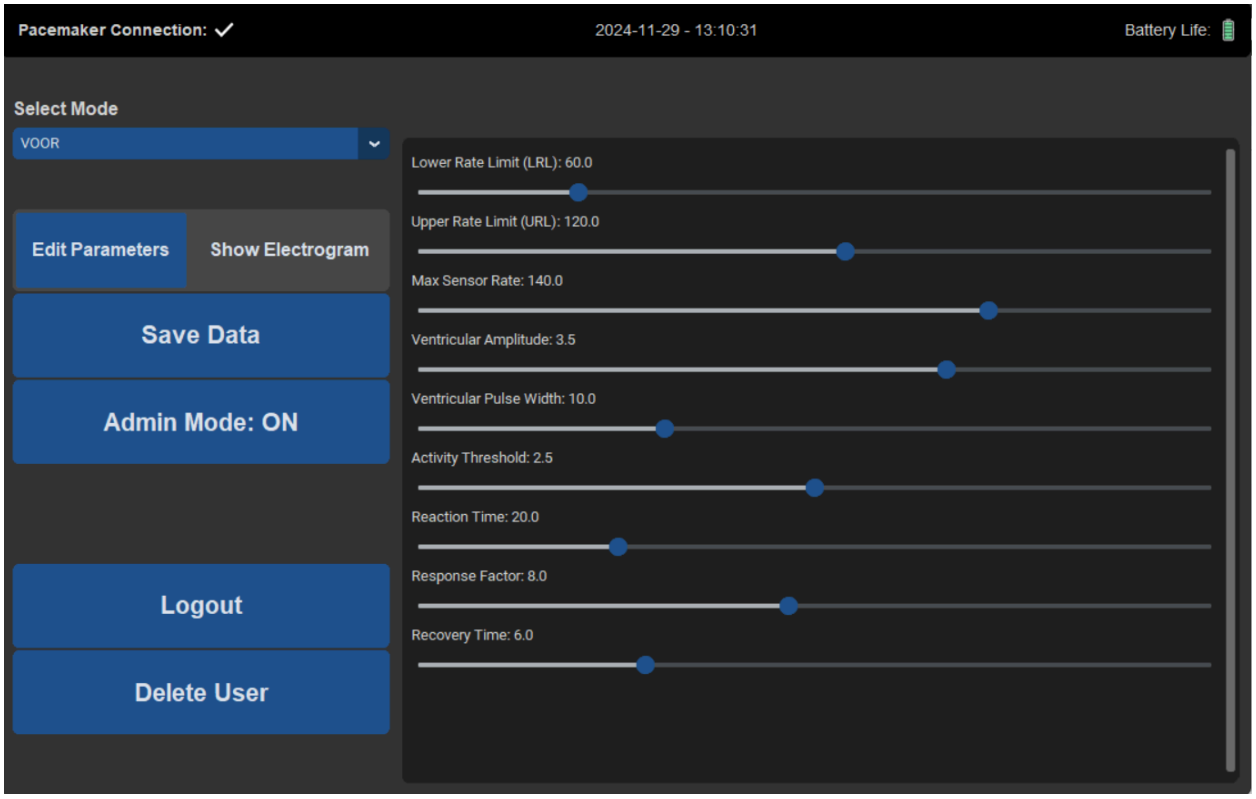


3.8.7.4.v2 Main Page

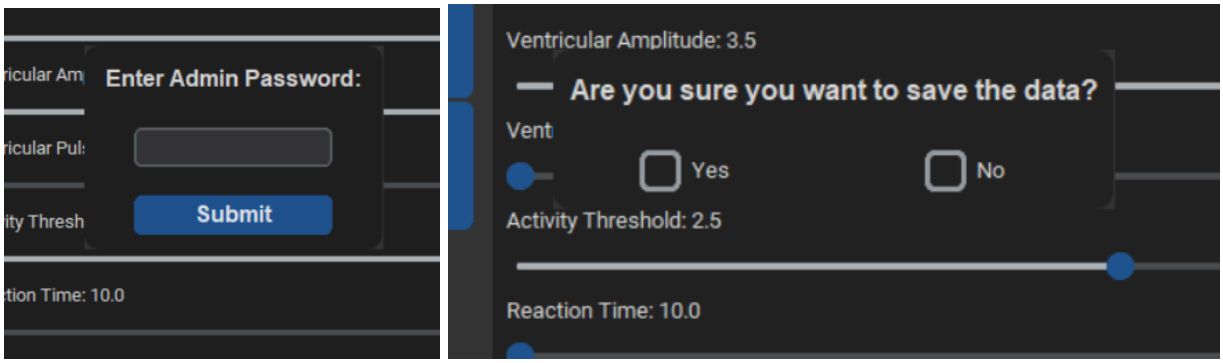
The Main Page is divided into three sections: informative banner at the top, interactive buttons on the left, and a parameter editor/electrogram graph on the right.



The Main Page is structured into three distinct sections to optimize usability. The top banner displays crucial information such as the time, battery life, and pacemaker connection status. The left panel houses interactive buttons, including a mode selector dropdown for AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, and VVIR, as well as buttons for saving data, logging in as an admin, logging out, and deleting the current user account.

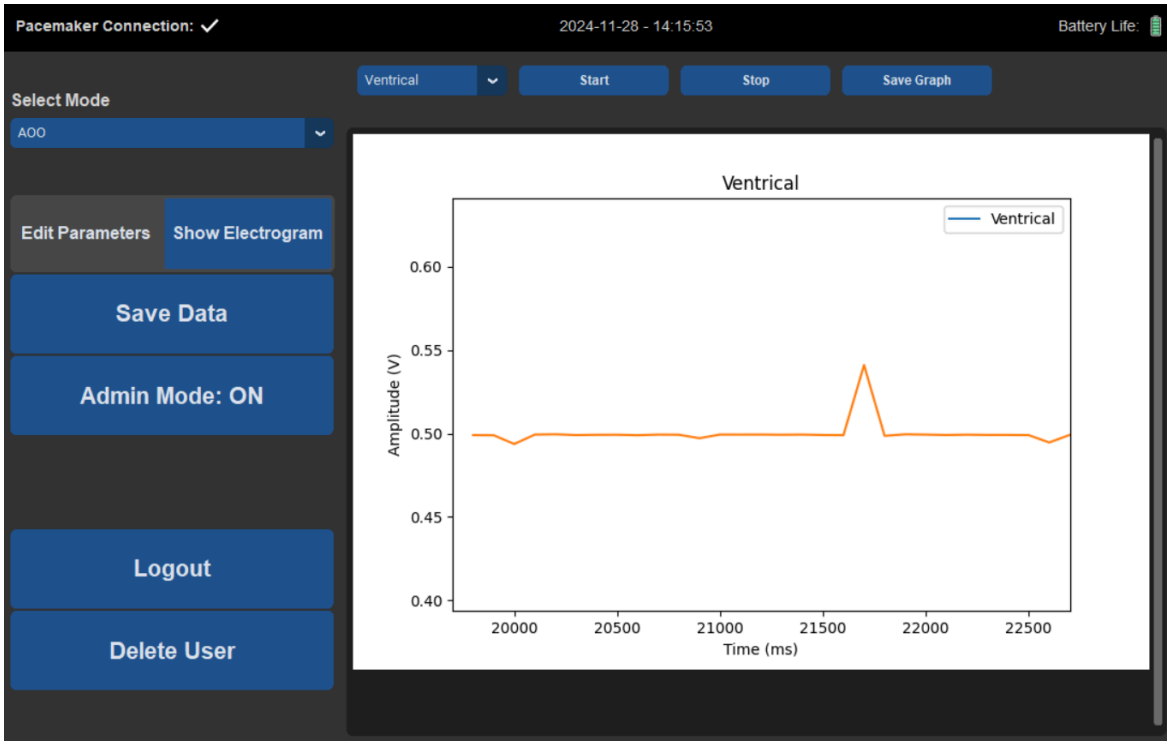


Access to sensitive functions, such as deleting user accounts or saving data, is restricted to admin users and requires a password for entry. Once in Admin Mode, users must confirm critical actions, such as deleting a user or sending data to the pacemaker, through additional prompts, ensuring accidental changes are avoided.



On the right side of the page, the interface dynamically switches between two modes: parameter editing and electrogram display. When in parameter editing mode, users can adjust settings using sliders, with intrinsic maximum ranges set for each parameter. The displayed values are initially loaded from the user's saved data for the selected mode. Conversely, in electrogram mode, the interface displays a real-time graph showing data exchanged between the DCM and the pacemaker. A dropdown menu allows users to choose whether to graph atrial

data, ventricular data, or both. Users can start, stop, and save the graph using the provided controls, offering flexibility in data visualization.



3.8.8.v2 Future DCM Design Decisions v2

Several improvements have been identified to enhance the functionality and user experience of the DCM. On the Create New User page, users should be required to confirm their passwords to ensure accuracy and awareness of their credentials. Password complexity should also be enforced by setting minimum length requirements and mandating the inclusion of special characters. Additionally, users should provide an email address during account creation, enabling password recovery and system notifications in case of issues with their account.

The login page could benefit from the addition of a password recovery feature, allowing users to retrieve forgotten credentials via their username and registered email. The battery life indicator should also be updated to display the pacemaker's actual battery status more accurately in real time.

On the Main Page, users should have the ability to customize aspects of the interface, such as the electrogram graph and the application's color scheme. Options for light mode and dark mode would improve accessibility and user comfort. Furthermore, user data should be stored in a way that facilitates access to historical records by doctors or administrators, ensuring comprehensive insights into patient settings and past performance data.

3.8.9 Version History

This section will briefly highlight changes in version history of the program from the beginning of the first saved file to what specific changes and design decisions were made with each new version. This will include details of classes, functions, and other design decisions from this area that were made, why they were made, and how it affects/benefits the program.

Version (MM/DD, all 2024)	**Notable Changes and Reasoning**
09/19, Original File Upload	Created GUIa1.py to be the main file that runs the program. Added functions to open a login page, create user page, and main page using tk and ctk. Added a text file called users.txt that stores user data and allows a user to log in or create a new user based on certain factors.
09/28	Changed the GUIa1.py file from just functions to classes with associated functions inside of them. Added UserManager class that could read users and save users. Added LoginPage class can create widgets for the page, handle login requests, allow a user to login, and direct you to the CreateUser page if you'd like to make a new user. Added CreateUser class that creates widgets for the page, handles requests to create a user, directs you back to the LoginPage class after successfully creating a user, and raises errors if any necessary parameters aren't achieved. Added MainPage class that creates widgets for the main page and allows the user to view data.
10/01	Added CustomTkinter library for better GUI aesthetics.
10/10	Added functions to the MainPage class to toggle and Admin mode setting based on a button press that allows parameters to be changed, can delete the current user based on a button press, show the electrogram on a graph, an edit frame option to edit parameters, and update the edit frame based on which variable is selected.
10/14	Added functions to the MainPage class that can check periodically if a microcontroller has established connection to the user, call a segment button that switches between the electrogram graph, update the label on the graph, and update the plot over time.
10/22, Assignment 1 Code	Commented majority of code to ensure straightforwardness and understanding. Fixed positioning of specific widgets to fit desired format.

	Cleaned up and organized classes and functions to ensure readability. Added the encrypt and decrypt functions to UserManager class. Adjusted the encryption key to ensure it will always encrypt passwords when a new user is created, and decrypt it when checking if a password entered matches a password in the text file when user logs in.
11/3	Implemented PySerial communication so DCM Python code could interact and exchange data with Simulink and Heartview. Edited functions and classes accordingly based on GUI modifications, serial communication functionality, and the addition of new parameters.
11/20	Adjusted electrogram graphing functionality to ensure graphs could be displayed for atrial, ventricular, or both at the same time. Added user database through json files and additional security for accounts.
11/26, Assignment 2 Code	Added modularity to code by splitting up classes into specific files, having a main file, and additional folders for databases and graphs to ensure easy accessibility, simplicity, and strong organization. Finalized PySerial communication code and attached file for communication and display on the electrogram graph in the code.

3.9 DCM Validation and Verification

3.9.1 Verification

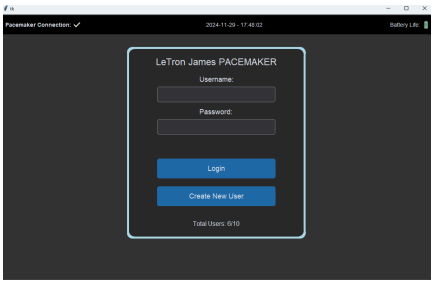
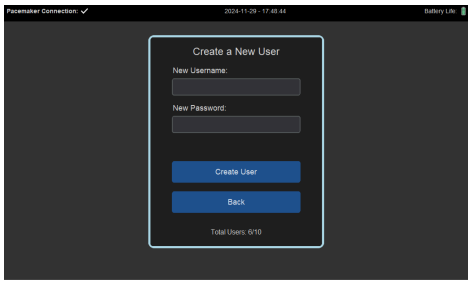
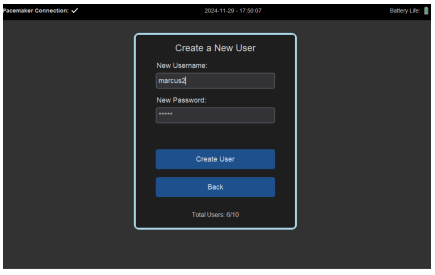
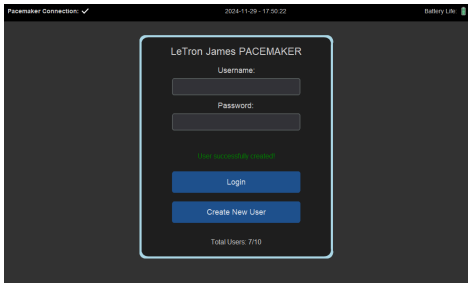
The purpose of DCM verification is the testing of the system in a non-operational state to confirm requirements and design based on code reviews, walkthroughs, and demos. The main focus of this section is looking at the program to determine how specific code, inputs, and GUI display match the purpose and goals of what needed to be achieved by the DCM at this stage.

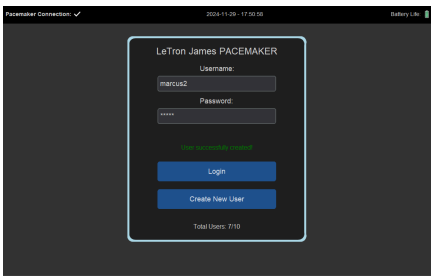
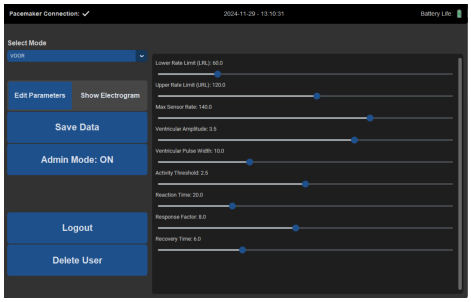
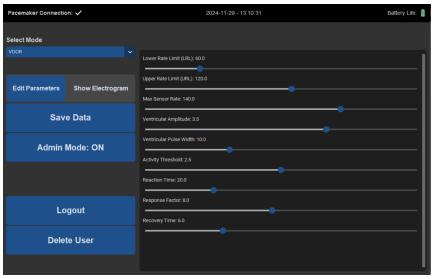
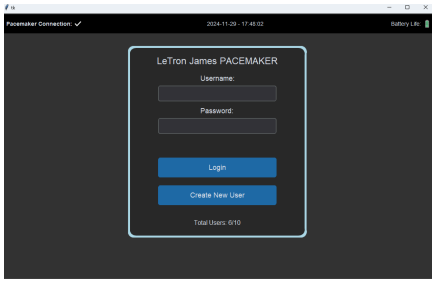
3.9.1.1 Windows

As shown in **3.8 DCM Design Decisions**, the GUI has working windows that display what the user can interact with based on whether they are at the login page, create user page, or main page.

3.9.1.1.v2 Windows

As per **3.9.1.1 Windows**, none of the functionality of any windows have been changed, and no other windows have been added. The GUI begins with having you at the login page, where you can go to the create user page to create a new user, and/or login back on the login page with a user which will take you to the main page.

Test	Before	After	Pass/ Fail
Login to CreateUser (press CreateUser button)			Pass
CreateUser to Login (press either button)			Pass

Login to Main (press Login button)			Pass
Main to Login (press Logout button)			Pass

3.9.1.2 Widgets

Widgets have all been correctly implemented to align with the requirements and design decisions.
The right frame correctly changes to represent either the mode parameters or electrogram data. Based on the chosen mode from the segmented button.

```
        self.segmented_button = ctk.CTkSegmentedButton(self.master,
values=["Edit Parameters", "Show Electrogram"],
command=self.segment_button_callback)
```

The admin button has been programmed to only allow the user to change the parameters for the chosen mode when it has been activated.

```
        admin_mode_button = ctk.CTkButton(self.master, text="Admin Mode: OFF",
command=self.toggle_admin_mode)
```

3.9.1.2.v2 Widgets

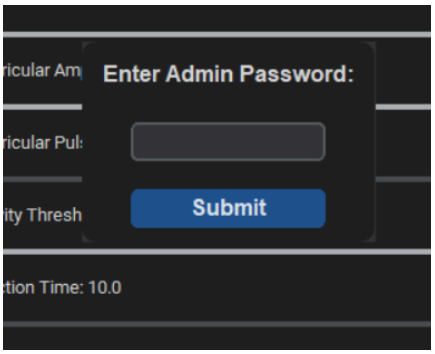
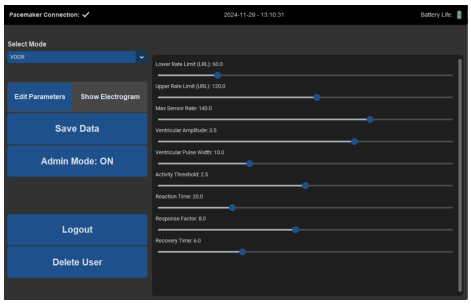
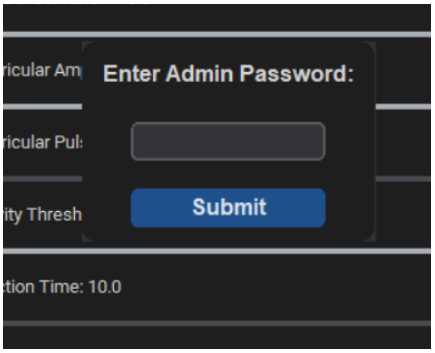
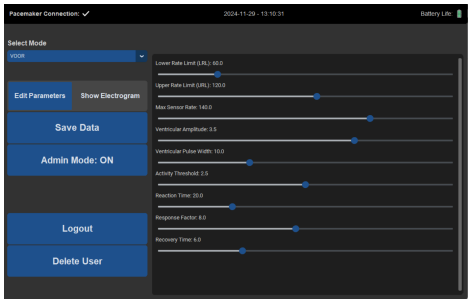
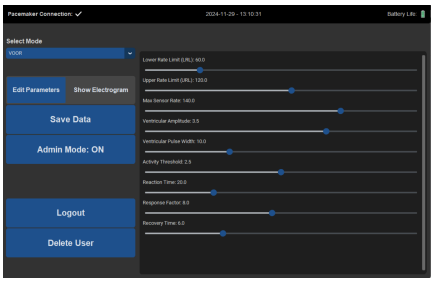
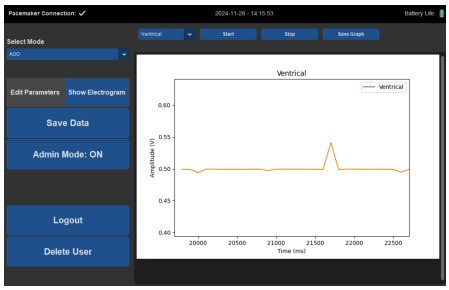
Widgets have been adjusted and manipulated according to the new requirements and design decisions.

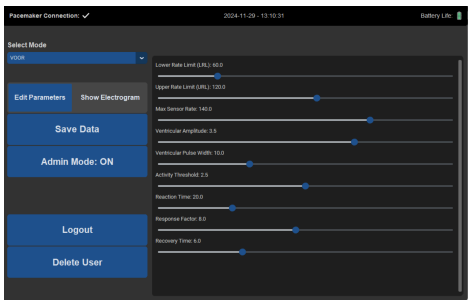
The right frame still has responsibility to change either the mode parameters or electrogram data based on the chosen mode from the segmented button.

```
def segment_button_callback(self, value):
        if value == "Show Electrogram":

            self.show_electrogram() # Call the function to show the electrogram
frame
            self.reset_plot() # Reset the plot when electrogram is shown

        elif value == "Edit Parameters":
            self.show_edit_frame() # Call the function to show the parameter
editing frame
```

The admin button has the same functionality, but now prompts the user for a password in order to access the admin perks. If the password is incorrect, the user will not get access and a pop up will display. Vice versa will happen for a correct password.

```
if not self.admin_mode.get():  # Admin Mode is OFF, prompt for a password
            # Create a popup frame and store it as an instance attribute
            self.popup_frame = ctk.CTkFrame(self.master, corner_radius=10)
            self.popup_frame.place(relx=0.5, rely=0.5, anchor="center")  # Center
the popup frame
else:
            # Incorrect password: disable admin mode and show an error message
            self.admin_mode.set(False)
            self.admin_mode_button.configure(text="Admin Mode: OFF")
            self.edit_data_button.configure(state="disabled")
            self.delete_user_button.configure(state="disabled")
            self.update_edit_frame(self.initial_state.get())
```

Test	Before	After	Pass/ Fail
Admin Button Success			Pass
Admin Button Fail			Pass
Show Electrogram			Pass

Show Edit Parameters			Pass

3.9.1.3 Parameters

Modes on the main page, AOO, VOO, AAI, and VVI are selected via a dropdown menu. Each mode has a collection of adjustable parameters. The adjustable parameters can be modified using a slidable bar when in admin. The slidable bar enforces an upper and lower bound for each parameter.

```
("Lower Rate Limit (LRL)", 30, 180, self.lower_rate_limit, 5)
```

Example of slider parameters. Has range between 30-180, updates variable lower_rate_limit, slider increments in 5s.

3.9.1.3.v2 Parameters

Added modes on the main page, AOOR, VOOR, AAIR, and VVIR, can also be selected via the dropdown menu. These modes also have their own unique collection of adjustable parameters. They can still be modified using the slidable bar when in admin. The slidable bar still enforces an upper and lower bound for each parameter.

3.9.1.4 Security

The passwords that the new user creates are securely encrypted through the use of an external security key and saved into the user.txt file.

```
def _encrypt_password(self, password):
        """Encrypt the password."""
        return self.cipher.encrypt(password.encode()).decode()
```

The user gets notified what error they are encountering when inputting an incorrect username or password when creating a new user.

```
if new_username in users:
            self.show_error("Error: Username already exists! Choose a different
one.")
        elif len(users) >= 10:
```

```
            self.show_error("Error: Max Number of users reached.")
        elif " " in new_username or ":" in new_username or " " in new_password
or ":" in new_password:
            self.show_error("Error: Usernames and passwords cannot contain
spaces or colons.")
        elif "" == new_username or "" == new_password:
            self.show_error("Must create both a username and password.")
        else:
            self.user_manager.save_user(new_username, new_password)
            self.app.open_login_page(success_message=True)  # Show success
message on login page
```

The user gets correctly notified if they enter an incorrect username or password.

```
self.master.after(100, lambda:
self.login_error_label.configure(text="Incorrect username or password.",
fg_color="red"))
```

3.9.1.4.v2 Security

The existing security measures from Assignment 1 are still in the current implementation of the DCM for Assignment 2, with additional measures also being added.

The users data is stored safely in a JSON file within the database of the users folder, having encryption in their username and file location for additional security measures.

```
def _sanitize_username(self, username):
        """Sanitize the username to create a safe filename."""
        return re.sub(r'[^a-zA-Z0-9_-]', '_', username)
def _get_user_file_path(self, username):
        """Get the file path for the user's JSON file."""
        sanitized_username = self._sanitize_username(username)
        return os.path.join(self.user_dir, f"{sanitized_username}.json")
```

There is a password required to access admin mode so someone can't get into it through the user's computer unless they know what it is.

```
def check_admin_password(self):
        # Get the entered password
        entered_password = self.admin_password_entry.get()
```

The user is given a check box that requires them to confirm if they want to delete their user in order to prevent them from accidentally pressing the button and having no confirmation message leaving the user accidentally deleted.

```
def delete_current_user_check(self):
         # Create a popup frame and store it as an instance attribute
        self.popup_frame = ctk.CTkFrame(self.master, corner_radius=10)
        self.popup_frame.place(relx=0.5, rely=0.5, anchor="center")  # Center the
popup frame
```

The user is given a check box that requires them to confirm if they want to save their newly changed user data in order to prevent them from accidentally pressing the button and having no confirmation message leaving the user's data accidentally modified.

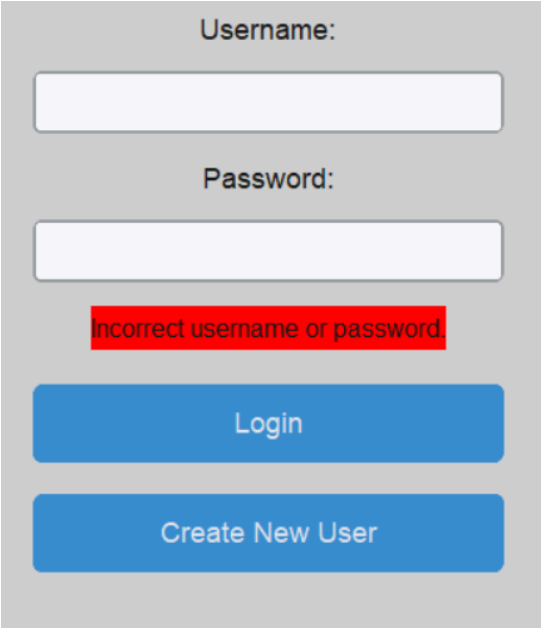
```
def update_user_data_check(self):
         # Create a popup frame and store it as an instance attribute
        self.popup_frame = ctk.CTkFrame(self.master, corner_radius=10)
        self.popup_frame.place(relx=0.5, rely=0.5, anchor="center")  # Center the
popup frame
```

3.9.2 Validation

The purpose of DCM validation is testing the software in an operational state to confirm outputs and system functionality. The main focus of this section is looking at the program and user interface to determine that the program is functioning the way it is required too, and allows for the user to be able to interact with the program as expected (this will be done with test cases that haven't already been talked about)

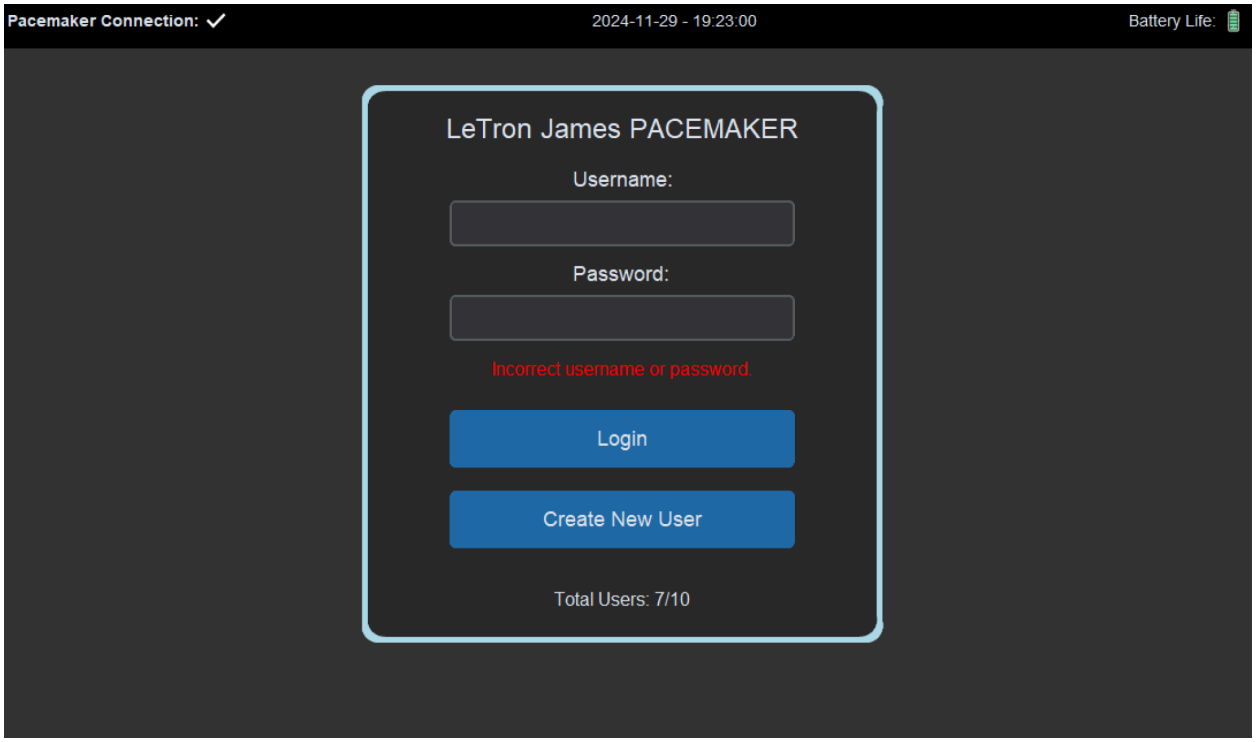
3.9.2.1.v1 Login Page

When an incorrect username or password is typed in, the login window prompts an error message in red letting the user know that they cannot log in.



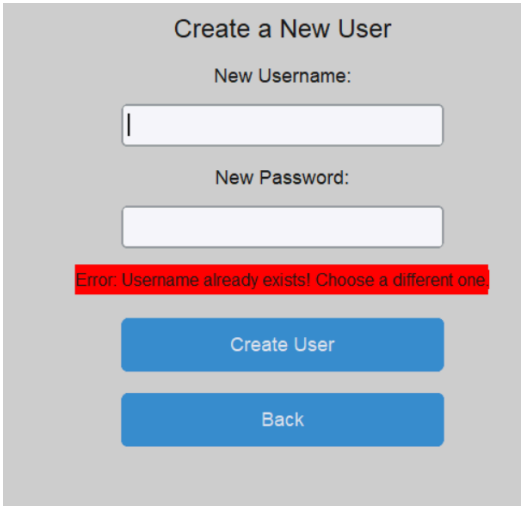
3.9.2.1.v2 Login Page

When an incorrect username or password is typed in, the login window prompts an error message in red letting the user know that they cannot log in.



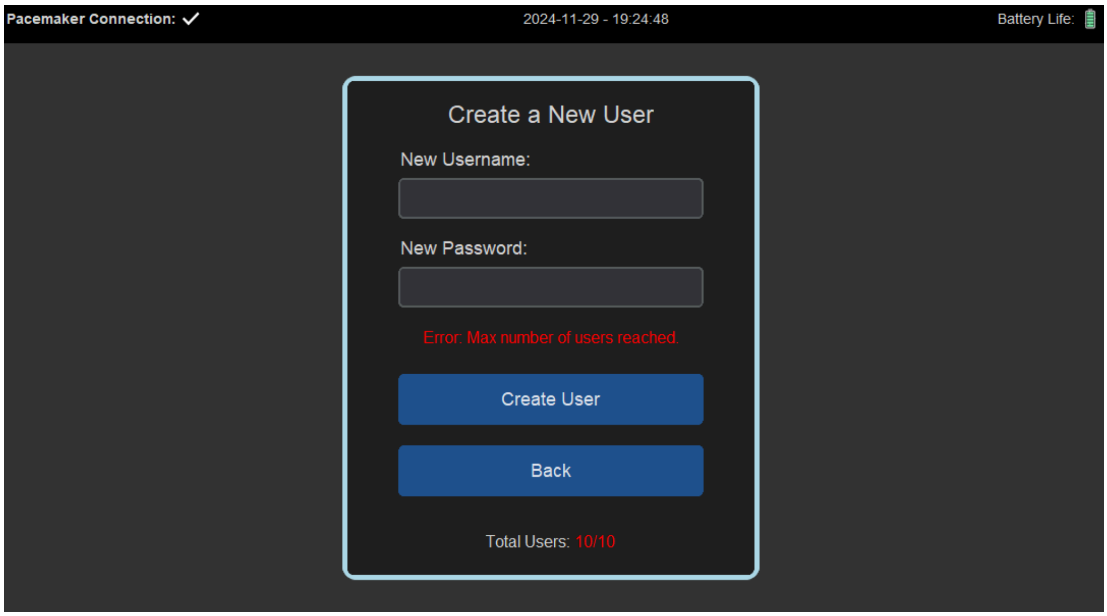
3.9.2.2.v1 Create User Page

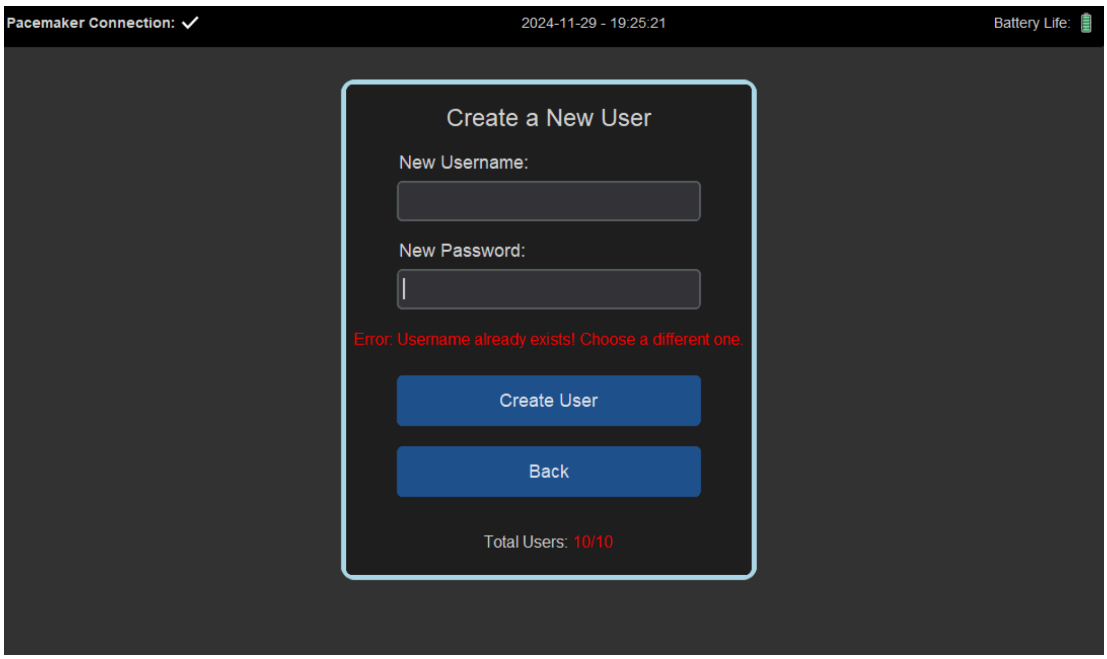
If a user tries creating an account with a username that already exists, an error message in red will show informing them of the issue. This ensures that no users have the same username, and avoids miscommunication or accidentally saving data into a wrong account. This same idea is established if a user tries creating a password that does not satisfy the requirements, or if 10 users are already in the text file. When the max amount of users has been reached for the database, the window prompts an error message in red letting the user know that there is a max amount of users reached.



3.9.2.2.v2 Create User Page

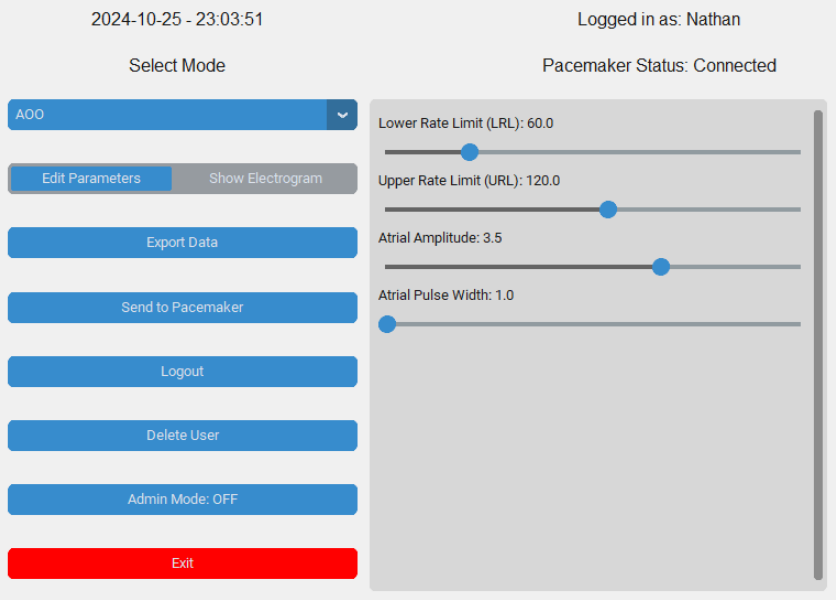
If a user tries creating an account with a username that already exists, an error message in red will show informing them of the issue. This ensures that no users have the same username, and avoids miscommunication or accidentally saving data into a wrong account. This same idea is established if a user tries creating a password that does not satisfy the requirements, or if 10 users are already in the text file. When the max amount of users has been reached for the database, the window prompts an error message in red letting the user know that there is a max amount of users reached.

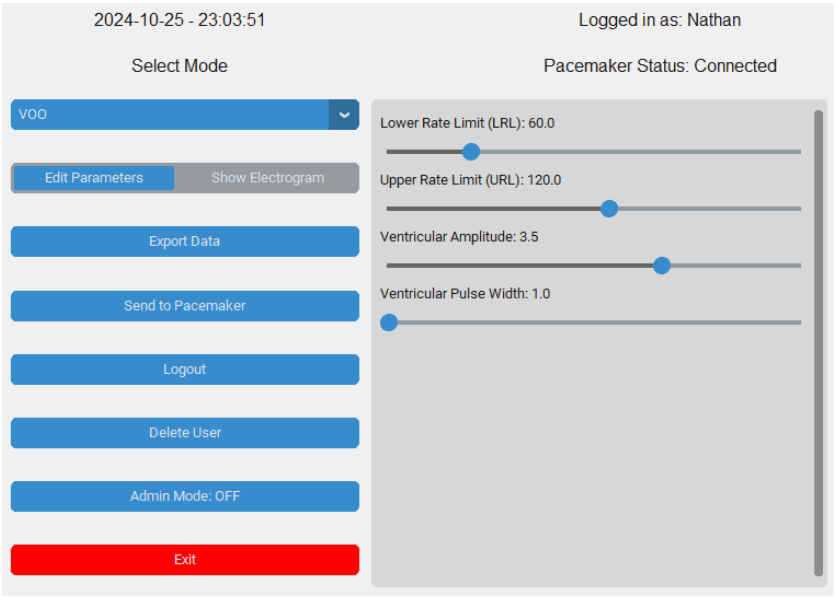


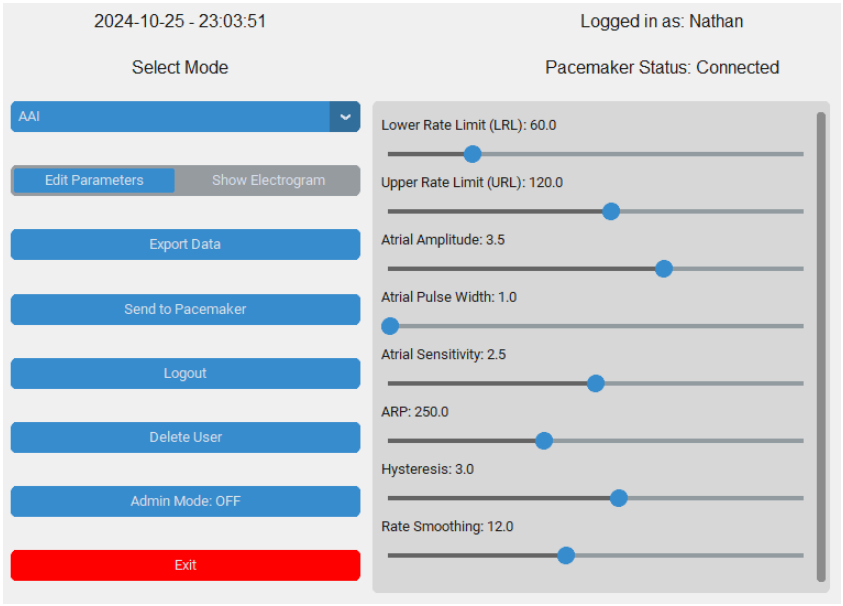


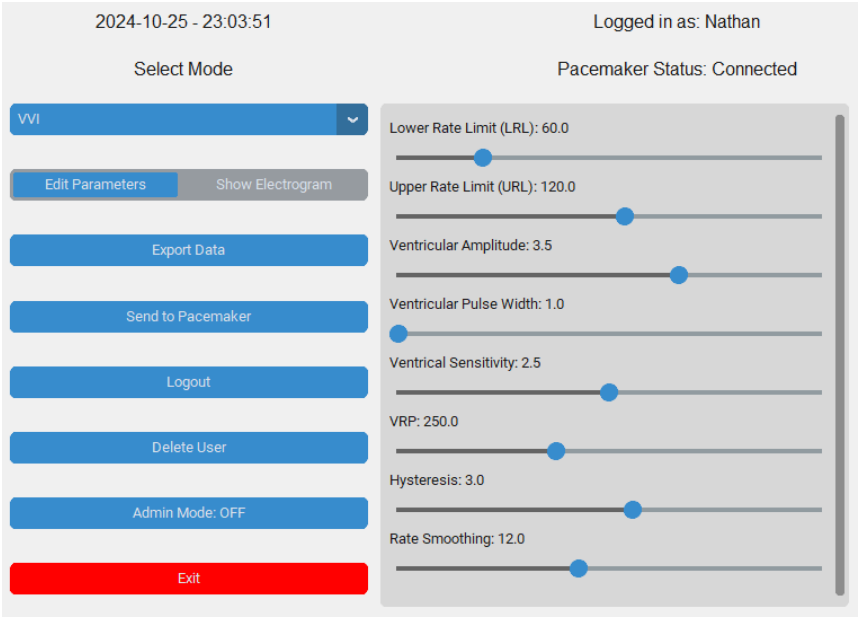
3.9.2.3.v1 Main Page

A user is restricted from editing any parameters unless Admin Mode is turned ON in order to promote safety and prevent accidents or a random person from changing parameters without permission. The sliders cannot be moved past their upper and lower ranges. Each mode has a unique set of parameters.



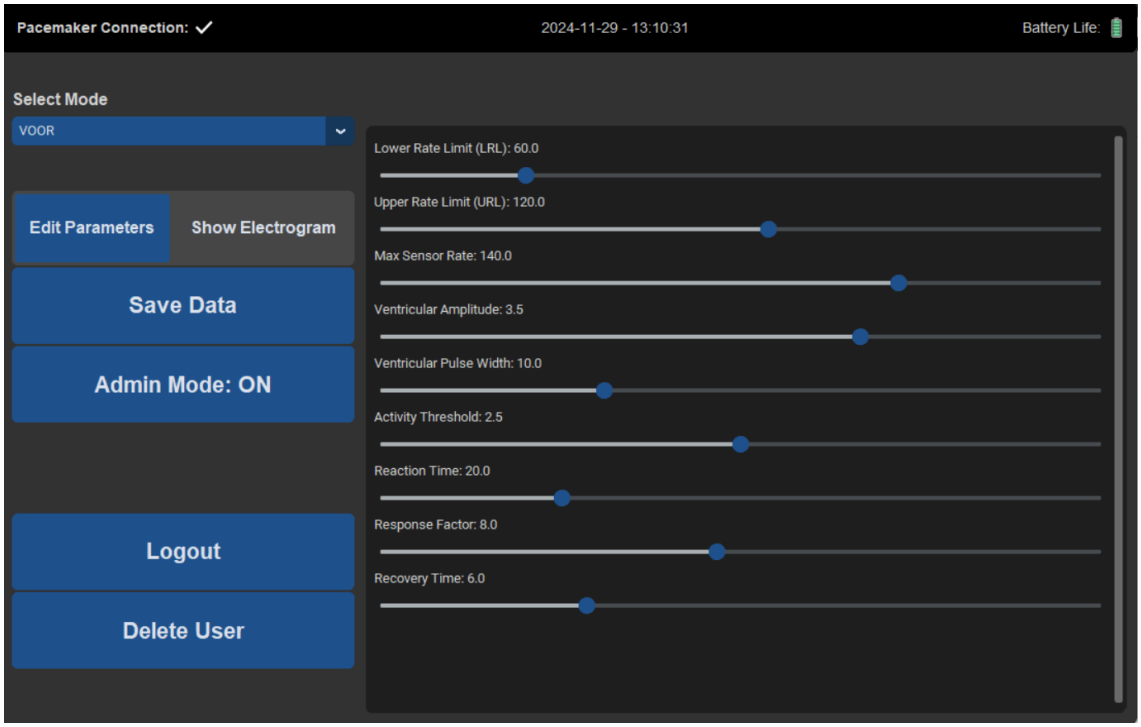


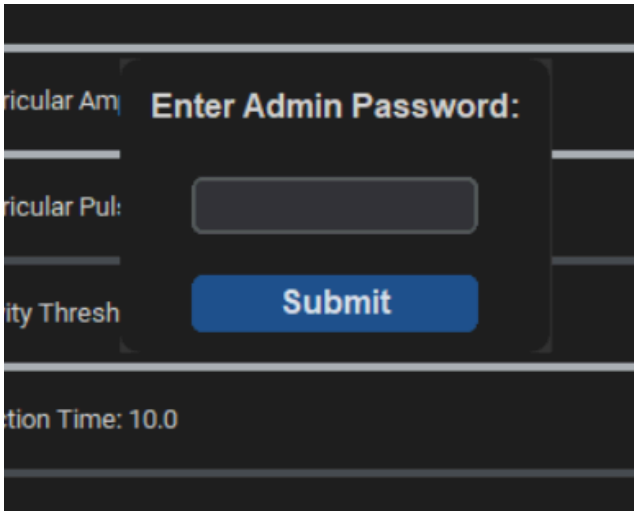

2024-10-25 - 23:03:51
Logged in as: Nathan
Select Mode
Pacemaker Status: Connected
AAI
Edit Parameters
Show Electrogram
Export Data
Send to Pacemaker
Logout
Delete User
Admin Mode: OFF
Exit
Lower Rate Limit (LRL): 60.0
Upper Rate Limit (URL): 120.0
Atrial Amplitude: 3.5
Atrial Pulse Width: 1.0
Atrial Sensitivity: 2.5
ARP: 250.0
Hysteresis: 3.0
Rate Smoothing: 12.0


2024-10-25 - 23:03:51
Logged in as: Nathan
Select Mode
Pacemaker Status: Connected
VVI
Edit Parameters
Show Electrogram
Export Data
Send to Pacemaker
Logout
Delete User
Admin Mode: OFF
Exit
Lower Rate Limit (LRL): 60.0
Upper Rate Limit (URL): 120.0
Ventricular Amplitude: 3.5
Ventricular Pulse Width: 1.0
Ventrical Sensitivity: 2.5
VRP: 250.0
Hysteresis: 3.0
Rate Smoothing: 12.0

3.9.2.3.v2 Main Page

A user is restricted from editing any parameters unless Admin Mode is turned ON in order to promote safety and prevent accidents or a random person from changing parameters without permission. In order to access Admin Mode, the user is required to enter a password and if it is not correct, they will not be given access. A pop up message will appear letting the user know whether they entered the correct password or an incorrect password. The sliders cannot be moved past their upper and lower ranges. Each mode has a unique set of parameters. These modes now include the additional 4 modes AOOR, VOOR, AAIR, and VVIR.





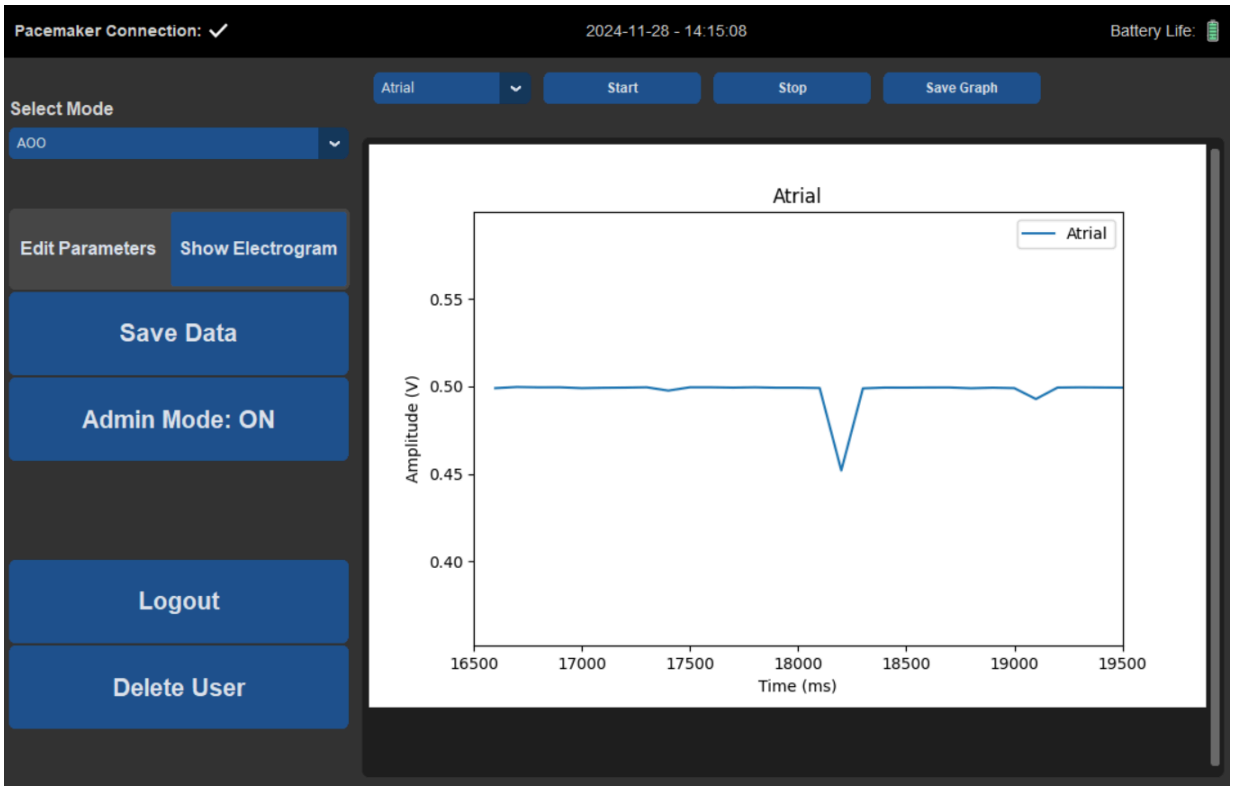
3.9.2.4.v2 Electrogram Graphing

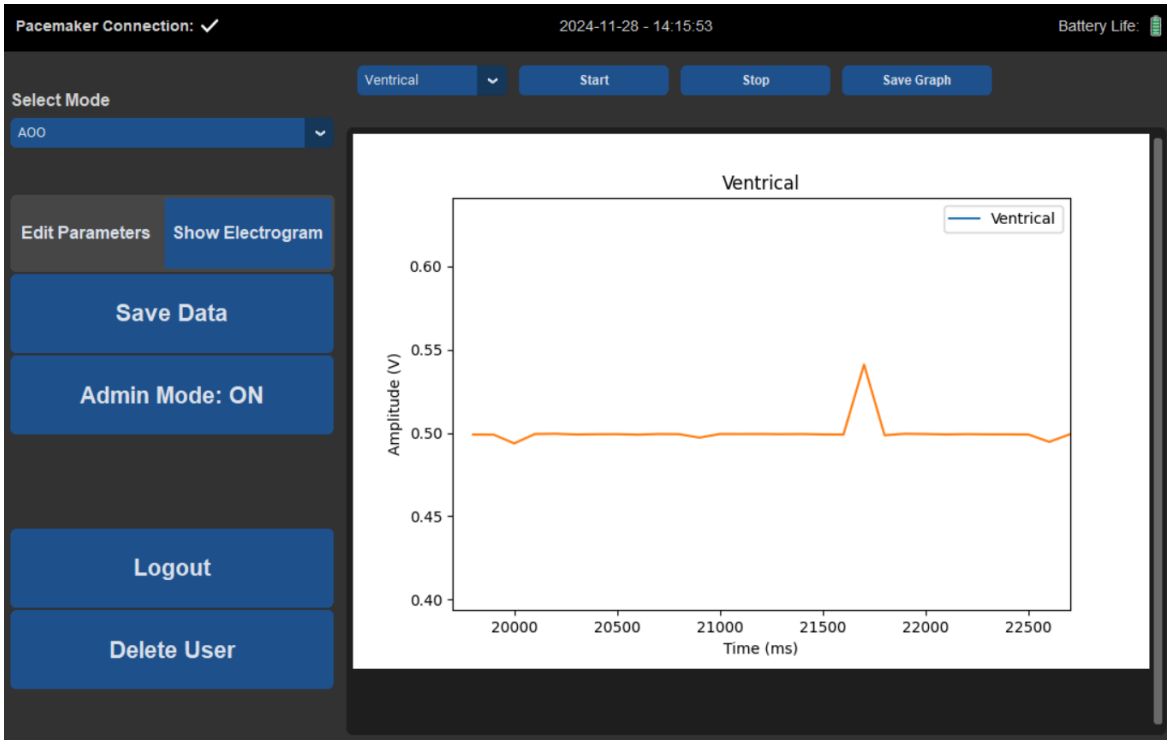
Electrograms (egrams) provide a graphical representation of the electrical activity within the heart chambers, offering insight into the user's cardiac functions.

The egram functionality in the project can be monitored in real time via the “Show Electrogram” button. The egram monitors the voltage at any given time in milliseconds relative to when graphing is started. The graph updates every 200 milliseconds.

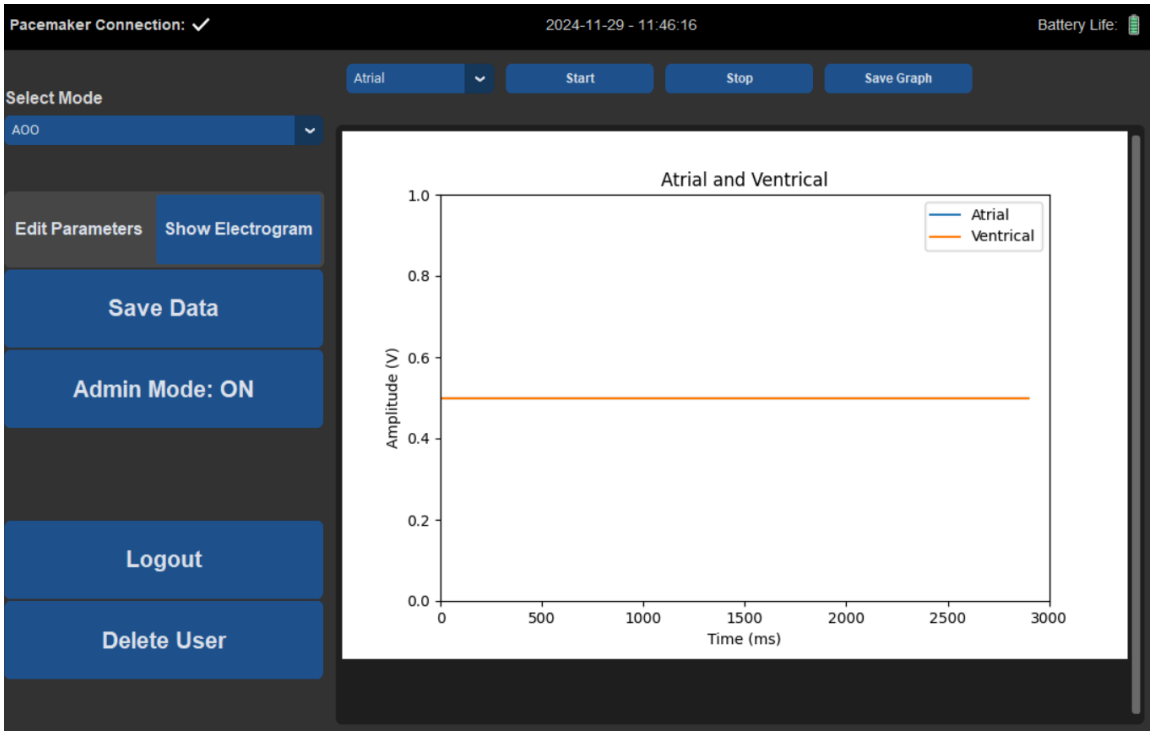
The egram has multiple interactive features including a drop down menu, and a “Start”, “Stop”, and “Save Graph” button.

The drop down feature allows the user to select between “Atrial” (displayed in blue), “Ventricle” (displayed in orange), and “Both” which display both chambers simultaneously. This feature is accessible while the graph is plotting live and while it is stopped. Upon launching the DCM the default option for the dropdown is “Atrial”.

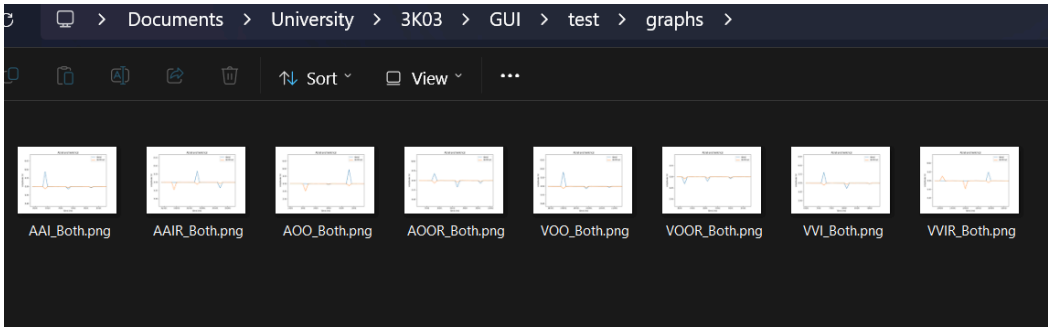




The Start, Stop, and Save Graph functionality intuitively allows the user to start, stop and save the graph. If the egram is started before new parameters are uploaded to the pacemaker the egram will graph the previous parameters. If there are no previous parameters the egram will not graph. The egram does not automatically start graphing and requires the user to initiate the graph with the “Start” button. Before started the graph looks as follows:



Once started the user can press the “Stop” button. This will cause the graph to stop updating. After stopping the plot the user is able to save the graph via the “Save Graph” feature. This feature will automatically save a file of the graph to the user's computer based on the location of the DCM files. The saved graphs are saved under different names to prevent graphs from being overridden. If the DCM does not have access to the file location where the graphs are saved the DCM will return an error. Additionally when paused the user can start the graph again which will cause the real time graph to resume.

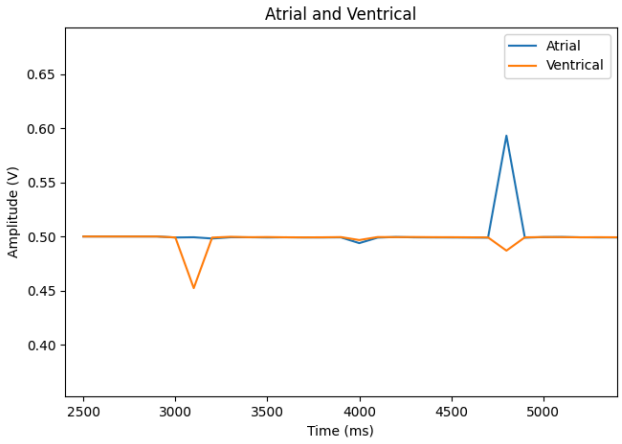
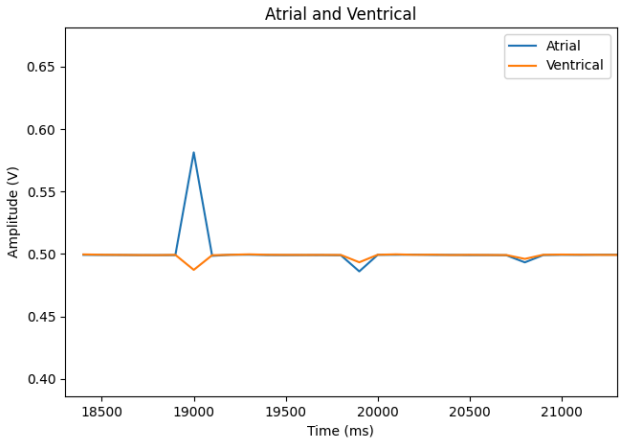


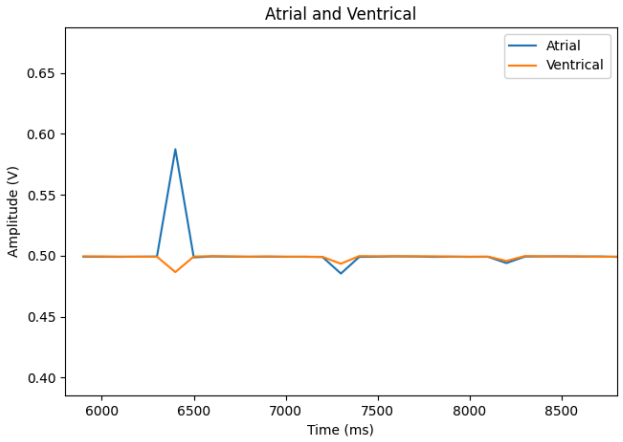
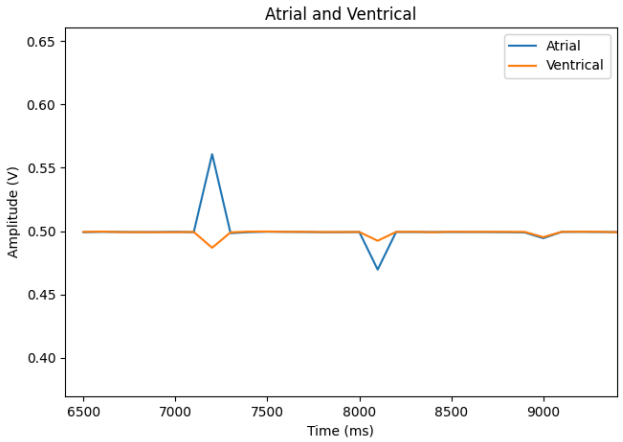
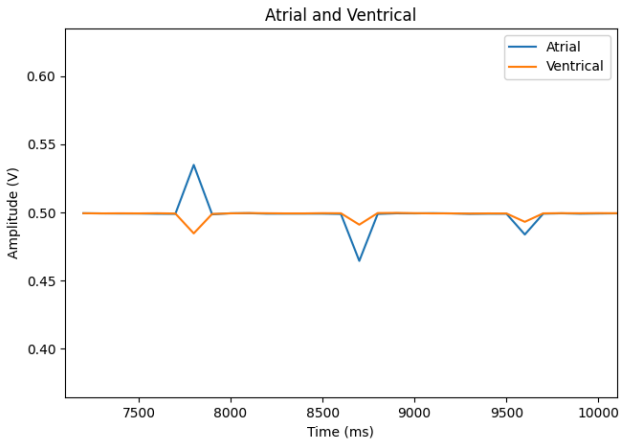
Both the standard user and admin are able to access all functionality related to the egram.

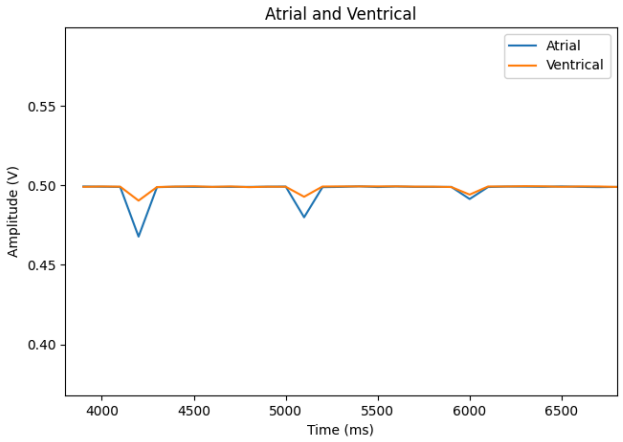
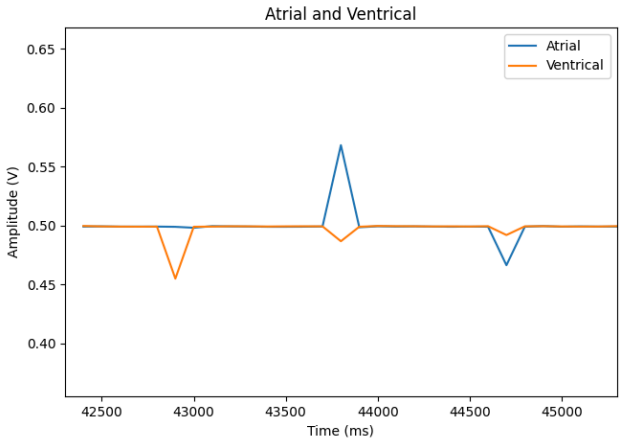
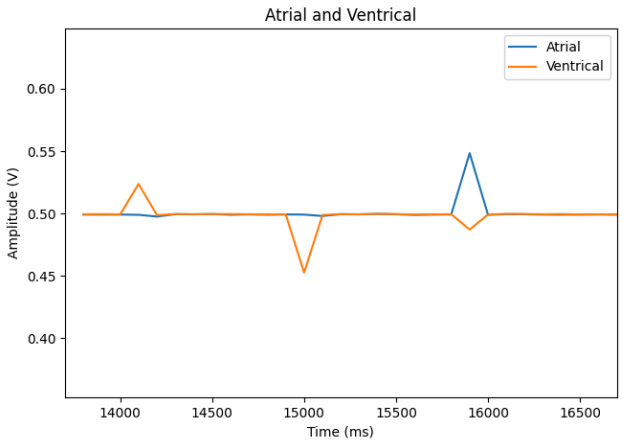
Egram data is collected via serial communication between Simulink, the DCM and the pacemaker. Egram data is verified via a parameter verification. The parameter verification compares the sent parameters to the received parameters, which are expected to be the same.

The egram is plotted using floats received from Simulink, these floats are from pins A0 (Atrial egram data) and A1 (Ventricle egram data).See section 2.5 for more details.

Testing

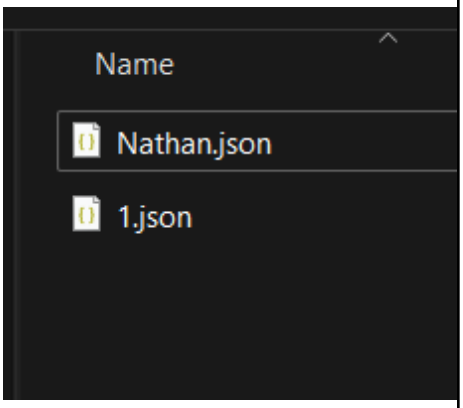
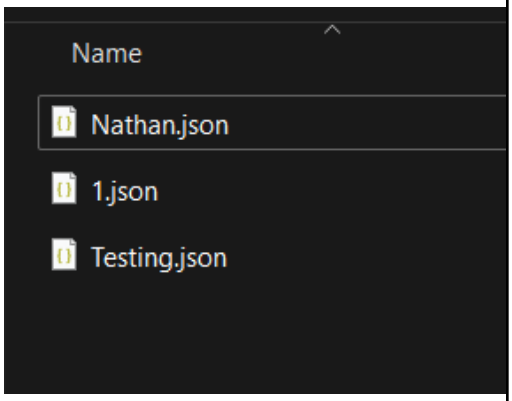
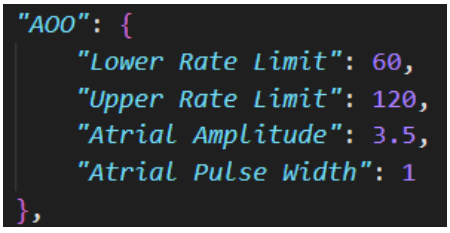
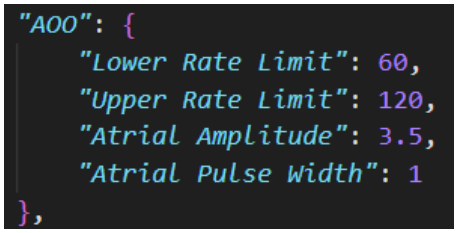
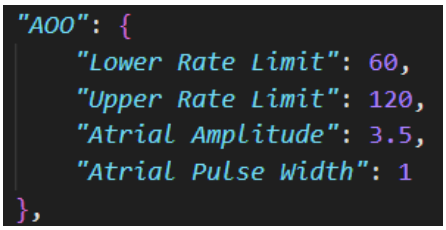
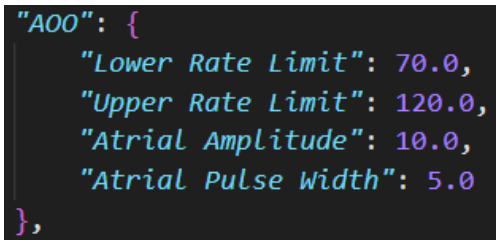
Mode	Output	Pass/Fail
AOO		Pass
VOO		Pass

AAI		Pass
VVI		Pass
AOOR		Pass

VOOR		Pass
AAIR		Pass
VVIR		Pass

3.9.2.5 Users Database

The user Database is meant to store the users previous inputs. This is achieved by reading and modifying external files. Each user has their own external file to prevent cross over. Within the users database is their username, encrypted password, and variables for all parameters given any mode. Variables for parameters are determined by what mode you are in, not what the parameter is, for example if atrial pulse width is 1 in AOO it can be different in other modes such as AOOR. Given a new user is created a new database will also be created and be populated with the default parameters.

Test	Before/Expected	After/Actual	Pass/ Fail
Create New User			Pass
Default Variables			Pass
Modify Variables			Pass
Password Encryption	Hashed Password*	"password": "gAAAAABnRzagthglVrj2Ggn-3MPDy04CvyGVWta8kC87y3biW6kYTKHWmGuJHHyjPECwz7xGw5_F1UMYmh9oIjuwY3PFpt_jSA=="	Pass

4. Assurance Case

The pacemaker is a safety critical medical device which can be categorized as a “Class C” device under the IEC 62304 standard because a possible software failure can result in serious injury or death to the patient. Therefore, conducting a comprehensive hazard analysis is going to be critical for developing a valid assurance case.

4.1 Hazard Analysis

In order conduct a comprehensive hazard analysis, a Failure Mode and Effects Analysis (FMEA) was conducted and is shown below:

Table: FMEA of pacemaker and DCM

Design Functions	**Failure Mode**	**Effects of Failure**	**Causes of Failure**	**Detection**	**Controls**
AOO pacing mode	Does not pace the atrium when required	Serious complication and/or death of the patient	a. Mode selector state failure	heartview graphs	Confirmation checks
			b. Incorrect mode selected by the user	Simulink graphs	
			c. Incorrect order of steps for setting the charge and discharge state	truth tables	
			d. Incorrect mapping of state variables to hardware pins		
	Paces the atrium but PWM does not have enough energy	Inadequate response from the atrium which could cause complications	a. A_AMP is not high enough		
	Paces the atirum but at incorrect time intervals	Inadequate response from the atrium which could cause complications	a. LRL is incorrect		
VOO pacing	same as AOO	same as AOO	Same as AOO		

mode	but for ventricle	but for ventricle	but for ventricle		
AAI pacing mode	fails to sense atrial heartbeat accurately	Could pace too much or not pace when required	a. Sensing state failure		
			b. Incorrect mapping of state variables to hardware pins		
	does not pace the atrium at correct time intervals	Inadequate response from the atrium which could cause complications	a. LRL is incorrect		
			b. APW is incorrect		
			c. ARP is incorrect		
	paces the atrium with low energy PWM	Inadequate response from the atrium which could cause complications	a. A_AMP is not high enough		
VVI pacing mode	same as AAI but for ventricle	same as AAI but for ventricle	same as AAI but for ventricle		
AOOR pacing mode	Incorrect sensing of activity level	Inadequate or too much pacing for the current user activity level	a. incorrect mapping of hardware accelerometer data to state variables		
			b. inaccurate calculation of activity level using accelerometer data		
	incorrect rate adaptive pacing of atrium	Inadequate or too much pacing for the current user activity level	a. incorrect RATE calculation		

			b. incorrect APW calculation		
	Does not pace atrium when required	Serious complication and/or death of patient	a. Mode selector state failure		
VOOR pacing mode	Same as AOOR but for ventricle	Same as AOOR but for ventricle	Same as AOOR but for ventricle		
AAIR pacing mode	fails to sense atrial heartbeat accurately	Could pace too much or not pace when required	a. Sensing state failure		
			b. Incorrect mapping of state variables to hardware pins		
	does not pace the atrium at correct time intervals	Inadequate response from the atrium which could cause complications	a. LRL is incorrect		
			b. APW is incorrect		
			c. ARP is incorrect		
	paces the atrium with low energy PWM	Inadequate response from the atrium which could cause complications	a. A_AMP is not high enough		
	incorrect rate adaptive pacing of atrium	Inadequate or too much pacing for the current user activity level	a. incorrect RATE calculation		
			b. incorrect APW value		
			c. incorrect ARP value		
VVIR pacing mode	Same as AAIR but for ventricle	Same as AAIR but for ventricle	Same as AAIR but for ventricle		
Changing saved	Unintended or	The patient can	a. malicious	user testing	Admin

settings	malicious user can make changes to critical saved settings	receive incorrect response from the pacemaker which can greatly endanger their life	user gains access to saved settings		password is required
			b. software defect		
selecting critical parameters	Admin selects an option they did not want to	The patient can receive incorrect response from the pacemaker which can greatly endanger their life	b. lack of confirmation checks		Confirmation checks
serial communication	Data integrity is compromised in transit	The pacemaker can have unpredictable behaviour which can greatly endanger the patient's life.	a. lack of integrity checks		Data integrity checks
			b. Electromagnetic interference.		
Storing user information	safety of user data is compromised if a miswrite occurs to the main file	Incorrect user information can cause the pacemaker to deliver an incorrect response to the given user.	a. data breach		User data files are separate, encrypted passwords for accessing data
			b. data corruption		

In order to perform a more comprehensive hazard analysis, an STPA was also conducted to understand different unsafe control actions that the pacemaker can perform which will harm the user.

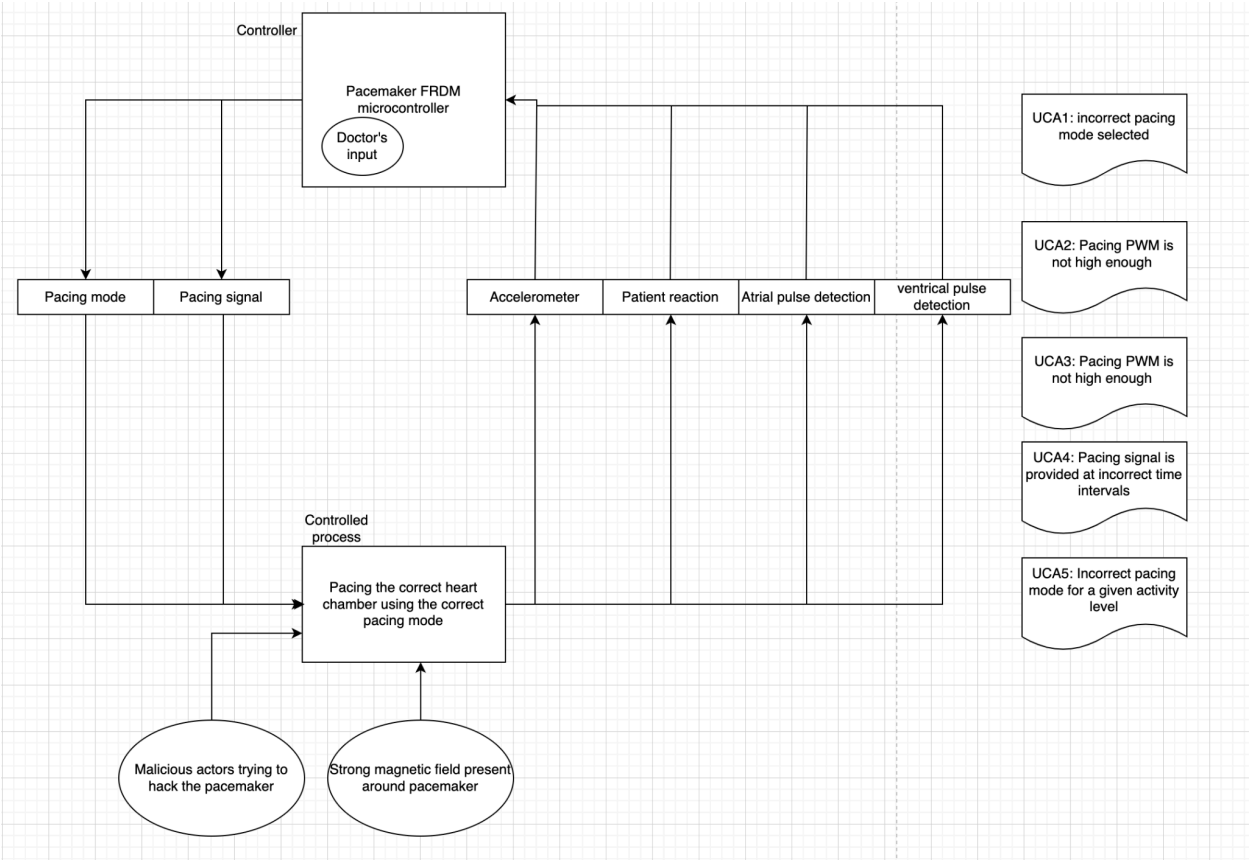


Figure: Pacemaker STPA

Based on the analysis done earlier, the following hazards have been identified:

Hazard Description	Hazard number
Incorrect pacing mode selected	H1
Pacing PWM is not high enough	H2
Pacing signal is provided at incorrect time	H3
Incorrect pacing rate for a given activity level	H4
Unintended user accessing saved settings	H5
Accidental selection and/or execution of commands by Admin	H6
Absence of data integrity when data is sent using serial communication	H7
Safety of user data from software glitches and malicious attempts to access them	H8

4.2 Goal Structuring Notation (GSN) assurance case for the Pacemaker