PACEMAKER PROJECT DOCUMENTATION

GROUP 25

Mohammed Fuzail, Tathagata Sikdar, Forbesii Du,
Ninad Thakker, Ryan Gowland, Teghveer Singh Ateliey

Introduction

Over the last 30 years cardiovascular disease (CVD) has seen a rise from 12.1 million to 20.5 million. In 2021 it was reported that leading cause of death worldwide was due to cardiovascular issues [1]. One type of cardiovascular disease is when the chambers of the heart do not pulse properly to sustain human life. John Hopps, a Canadian biomedical engineer invented the implantable cardiac pacemaker in 1998 [2].

A pacemaker is an electronic device that assists the body in making sure all chambers of the heart are paced at the proper rate to maintain a healthy human body. We were instructed to fabricate a pacemaker for this project. For our project we were given the following hardware and software to complete our task; hardware consisted of a NXP FRDM-K64F (Arduino) Board to act as the heart, another microcontroller to act as the pacemaker board all contained on one device to interact with efficiently. We were also given two sets of wire to connect the heart and the pacemaker to our devices. The software consisted of Heartview to debug and run testing on the heart/pacemaker, MATLAB R2020a and Simulink for the respective boards.

We utilized Simulink for code generation and the choice of programming language for the Device Control-Monitor (DCM) was given to us. More details regarding the Pacemaker (Simulink) and DCM components can be found in the upcoming sections.

Research and Planning

Pacemaker (Simulink)

Simulink software is something our team was not familiar with, due to this the phase of Research and Planning for the Simulink component consisted of familiarizing ourselves with Simulink and heartview. Simulink is a software tool offered by MATLAB, it offers to generate code from detailed state flow diagrams of the desired logic to be implemented. To utilize Simulink, one does not have to know coding but rather must describe the flow of the system one wants to build and translate it into a state flow diagram. To familiarize ourselves with Simulink several "hello world" type programs were created, making sure to experiment with different components as to get a good understanding of the software and how it worked. Many high-level state flow diagrams were created to blink light on the Arduino board. Along with getting accustomed to different components of Simulink individually, the tutorial documents provided were also referenced and a brief run-down of the Simulink documentations core topics was read through as a group. After this step and prior to any "coding" a good understanding of the functionality of the pacemaker, its different modes and safety considerations were brainstormed with the team. This meant going through many of the documents provided by the client (Course TA's) and making notes of key takeaways. Finally the group of six was divided into two groups of tree. For the first assignment Mohammed Fuzail, Tathagata Sikdar and Ninad Dhirenbhai Thakker were tasked with the DCM and Forbesii Du, Teghveer Ateliey and Ryan Gowland were tasked with the Pacemaker (Simulink). These groups switched roles for the second assignment.

Device Control-Monitor (DCM)

The Device Control-Monitor is best described as a guided user interface for interacting with the Pacemaker. The coding language in which it must be implemented was left for our team to decide. Our team first experimented with React, JavaScript, HTML and CSS initially but later switched to python.

Why Python?

Python is a versatile and popular programming language that offers several advantages for developing graphical user interfaces (GUIs) compared to web development technologies like React. One key benefit is the simplicity and readability of Python code, which allowed for faster development and easier maintenance of the GUI application. Additionally, Python boasts a rich ecosystem of GUI libraries, such as Tkinter, PyQt, and Kivy, providing our team with diverse tools for creating interactive and visually appealing interfaces. Furthermore, Python's cross-platform compatibility ensured that the GUI application could be seamlessly deployed on different operating systems without major modifications, enhancing the overall accessibility of the software. Coupled with the fact that not every member on our team was not familiar with React; Python was a compelling choice for GUI development in our team, especially since this project required rapid development, code clarity, and cross-platform support which are crucial considerations.

Requirements and Specifications

Below you will find the Requirements and the specifications along with unique names for each requirement for the pacemaker and DCM in sperate tables. In the Table you will notice a reference to the modes which may contain three or four letters. The legend for how to interpret these modes can be found below. This table and following information were sourced from the "PACEMAKER" and "Tutorial 4" documents provided by the client (Cours TA's)

	I	**II**	**III**	**IV (optional)**
Category	Chambers paced	Chambers Sensed	Response to sensing	Rate Modulation
Letters	O – None A – Atrium V – Ventricle D - Dual	O – None A – Atrium V – Ventricle D - Dual	O – None T – Triggered I – Inhibited D – Tracked	R – Rate modulation

Table 1: Bradycardia Operating Modes

Letter	**Explanation**

No response to sensing (O)	Pacing without sensing is asynchronous pacing. During asynchronous pacing, paces shall be delivered without regard to senses.
Triggered response to Sensing (T)	During triggered pacing, a sense in a chamber shall trigger an immediate pace in that chamber.
Inhibited response to Sensing (I)	During inhibited pacing, a sense in a chamber shall inhibit a pending pace in that chamber.
Tracked Response to Sensing (D)	During tracked pacing, and atrial sense shall cause a tracked ventricular pace after programmed AV delay, unless a ventricular sense was detected beforehand.
Rate Modulation (R)	During pacing, a custom pacing rate for the chambers must be determined based on data from accelerometer. NOTE: An accelerometer is a sensor which reports the acceleration in the x, y and z direction.

Table 2: Understanding Different Modes

For the following table of programable parameters, a green box indicates a parameter which is required for that mode and a red box indicates a parameter which is not required for that mode

	Mode	**AOO**	**AAI**	**VOO**	**VVI**	**AOOR**	**AAIR**	**VOOR**	**VIIR**
Parameter									
Lower Rate Limit									
Upper Rate Limit									
Maximum Sensor Rate									
Atrial Amplitude									
Ventricular Amplitude									
Atrial Pulse Width									
Ventricular pulse width									
Atrial sensitivity									
Ventricular Sensitivity									
VRP									
ARP									

PVARP									
Hysteresis									
Rate Smoothing									
Activity Threshold									
Reaction Time									
Response Factor									
Recovery Time									

Table 3: Programmable parameters for each mode

The following table contains the values for each of the aforementioned programmable parameters.

Parameter	Programmable Values	Increment	Nominal	Tolerance
Lower Rate Limit	30-50 ppm 50-90 ppm 90-175 ppm	5 ppm 1 ppm 5 ppm	60 ppm	±8 ms
Upper Rate Limit	50-175 ppm	5 ppm	120 ppm	±8 ms
Maximum Sensor Rate	50-175 ppm	5 ppm	120 ppm	±4 ms
Atrial Amplitude	0, .5-3.2V 3.5-7.0V	.1V .5V	3.5V	±12%
Ventricular Amplitude	0, .5-3.2V 3.5-7.0V	.1V .5V	3.5V	±12%
Atrial Pulse Width	0.05ms .1-1.9ms	-- .1 ms	.4 ms	.2 ms
Ventricular Pulse width	0.05ms .1-1.9ms	-- .1 ms	. 4 ms	.2 ms
Atrial Sensitivity	.25, .4, .75 1.0-10mV	-- .5mV	.75 mV	±20%
Ventricular Sensitivity	.25, .4, .75 1.0-10mV	-- .5mV	.5 mV	±20%
VRP	150-500 ms	10 ms	320 ms	±8 ms
ARP	150-500 ms	10 ms	250 ms	±8 ms
PVARP	150-500 ms	10 ms	250 ms	±8 ms
Hysteresis	Off or same choices as LRL	N/A	off	±8 ms
Rate Smoothing	Off, 3,6,9,12,13,15,18,21,25%	N/A	off	±1%

Activity Threshold	V-Low, Low, Med-Low, Med, Med-High, High, V-High	N/A	Med	N/A
Reaction Time	10-50 sec	10 sec	30 sec	±3 sec
Response Factor	1-16	1	8	N/A
Recovery Time	2-16 min	1 min	5 min	±30 sec

Table 4: Programmable Parameters Values

Pacemaker (Simulink)

Requirement ID	Requirement	Specifications
PACE-REQ-01	Must be able to emit an electrical stimulus.	Fundamentally the pacemaker should be able to pace and produce a pulse.
PACE-REQ-02	Must contain the following Modes utilizing all meaningful programmable parameters:	Meaningful programmable parameters for each mode can be found in the able above.
	AOO	Pace atrium, sense nothing and DON'T respond to sensing.
	VOO	Pace ventricle, sense nothing and DON'T respond to sensing.
	AAI	Pace atrium, sense atrium and inhibit as response to sensing.
	VVI	Pace ventricle, sense ventricle and inhibit as response to sensing.
	AOOR	Pace atrium with rate modulation, sense nothing and DON'T respond to sensing.
	VOOR	Pace ventricle with rate modulation, sense nothing and DON'T respond to sensing.
	AAIR	Pace atrium with rate modulation, sense atrium and inhibit as response to sensing.
	VVIR	Pace ventricle with rate modulation, sense ventricle and inhibit as response to sensing.

PACE-REQ-03	Rate Adaptive modes must track activity using the on-board accelerometer.	The pulse rate must consider the data from the accelerometer and the pulse rate (LRL) should increase once sufficient activity is detected.
PACE-REQ-04	Centralize Modes	All the modes must be integrated into one model.
PACE-REQ-05	Serial Communication	The programmable variables along with the modes should be pulled from the DCM after the user has specified the values and the DCM and the Simulink should be linked.
PACE-REQ-06	Pin Mapping	The state flow should not change if the pin map is altered, but rather the correct pin should map to its corresponding component.
PACE-REQ-07	Hardware Hiding	Abstract away the hardware from the design and make sure to implement modularity.
PACE-REQ-08	Safety Assurance	Develop respective amount of testing to validate design and mention an assurance case.

Table 5: Pacemaker Requirements

Device Control-Monitor (DCM)

Requirement ID	Requirement	Specifications
DCM-REQ-01	Desktop Application	Should be a desktop application and should support cross platform.
DCM-REQ-02	Include a Welcome Screen	Should have the ability to register a new user (name and password) and to login as an exiting user. A maximum of 10 uses should be allowed to be stored locally.
DCM-REQ-03	Capable of managing windows	User interface shall be capable of utilizing and managing windows for

		display of text and graphics.
DCM-REQ-04	Processing	User interface shall be capable of processing user positioning and input buttons.
DCM-REQ-05	Programmable Parameters	User interface shall be capable of displaying all programmable parameters for review and modification.
DCM-REQ-06	Serial Communication	User interface shall be capable of visually indicating when the DCM and the device are communicating.
DCM-REQ-07	Lost Connection	User interface shall be capable of visually indicating when telemetry is lost due to the device being out of range and noise.
DCM-REQ-08	Another Pacemaker	User interface shall be capable of visually indicating when a different PACEMAKER device is approached than was previously interrogated.
DCM-REQ-09	Pacing Modes	User interface shall be capable of visually indicating all of the pacing modes mentioned in the Pacemaker requirements to the user.
DCM-REQ-10	E-gram Data	User interface shall be capable of fetching and visually displaying e-gram data.

DCM-REQ-11	Modularity and hardware hiding.	User Interface design shall incorporate OOP principles such as modularity.

Table 6: DCM Requirements

Design, Specifications and Justification

General Workflow

The general workflow of our pacemaker is as follows. After installing and running the desktop pacemaker application the user is welcomed via the welcome screen and is prompted to either login or sign up and will do the respective task. At this moment the user will be presented to the official page for our pacemaker. Over here the user will be able to select a pacemaker to connect to via the drop-down menu found on the page. Once the pacemaker is selected the user can pick any of the official modes offered and also its programmable parameters. At this point the DCM will send these values to the Pacemaker/Simulink where the logic for the pacemaker is implemented. The Pacemaker will send the electrical signals to the respective nodes depending on the mode picked by the user and then it will send the e-gram data to the DCM. On the GUI the user can find a button labelled e-gram and when clicked it will display the e-gram data collected by the pacemaker. During any of the aforementioned process the user can change the modes and its respective programmable parameters and the change will reflect in the signal.

Pacemaker (Simulink)

The top-level pacemaker software model is composed of 5 main modules. These modules include the Serial Input, Activity Level Analysis, Main State Machine, Hardware Pin Output, and EGRAM Data output. The Serial Input module serves as a repository for programmable parameters input into the DCM. Transforming user input parameters into relevant variables for manipulation downstream data processing and saving them to onboard memory. The Activity Level Analysis module utilizes the onboard accelerometer data along with the saved values to determine the adapted heart rate for adaptive pacing modes (VOOR, AAIR, VVIR, AOOR). It plays a crucial role in the pacemaker's decision-making process by dynamically adjusting pacing rate based on patient's activity level. These parameters are taken in by the Main State Machine, which serves as the brain of the pacemaker, hosting informed decision making based on the Activity Level Analysis data, Serial Input modules and heart sensors to prepare the output signal for Hardware Pin Output module. This module takes the calculated values of output parameters and registers the digital signal to a hardware output pin. Finally, the EGRAM Data Output module continuously sends data including analog signals from the atrium and ventricle through serial communication to provide monitoring and analysis, contributing to a comprehensive understanding of the pacemaker's performance.

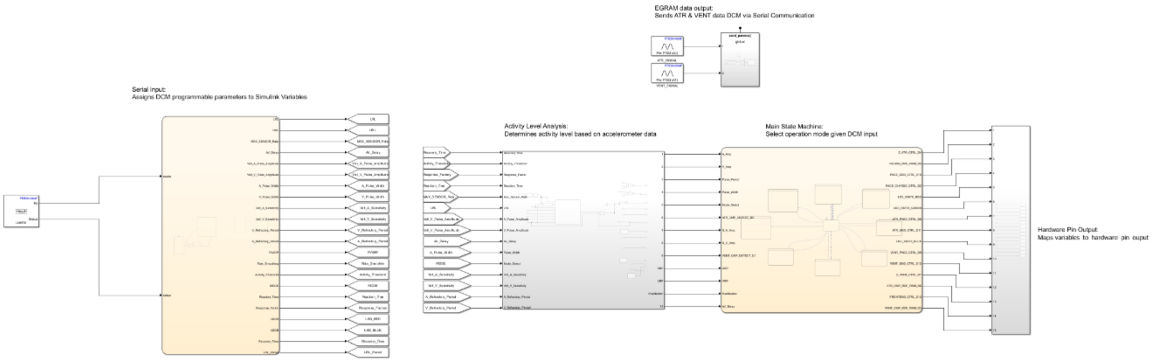


Figure 1: Top Level Pacemaker Architecture

Serial Input

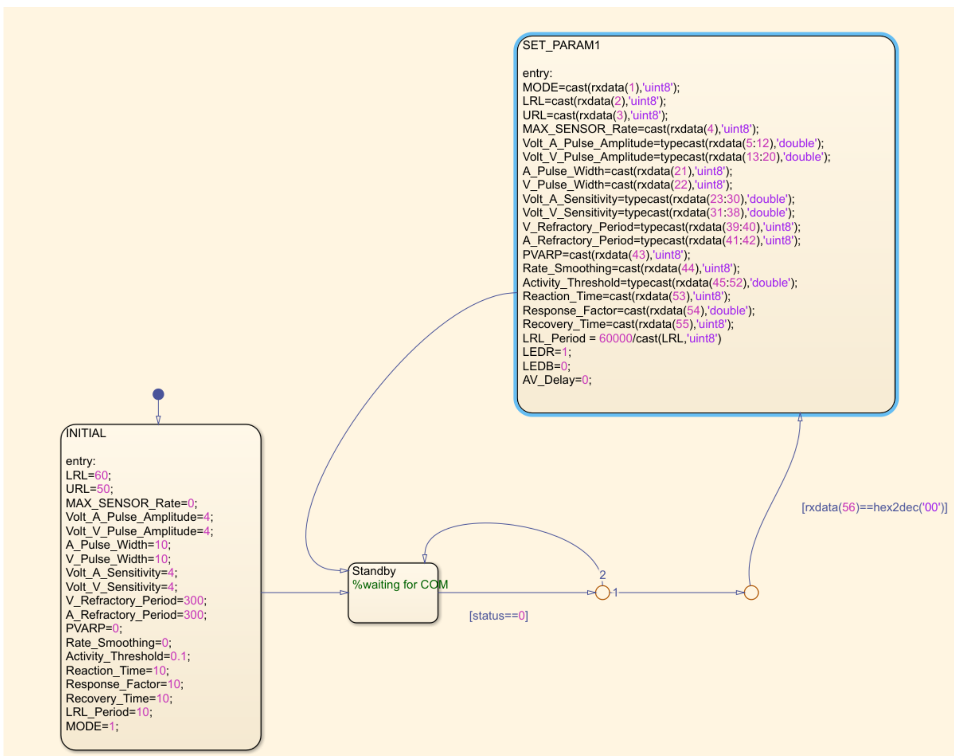


Figure 2: Serial Input Subsystem

The Serial Input module starts with a safe initialization of all variables. The variables indicated in figure 2 can be found with their description in the table below.

Variable Name	Description
LRL	Lower Rate Limit
URL	Upper Rate Limit
MAX_SENSOR_Rate	Maximum Sensor Rate

AV_Delay	Atrioventricular Delay
Volt_A_Pulse_Amplitude	Voltage Amplitude for Atrial Pulse
Volt_V_Pulse_Amplitude	Voltage Amplitude for Ventricular Pulse
A_Pulse_Width	Atrial Pulse Width
V_Pulse_Width	Ventricular Pulse Width
Volt_A_Sensitivity	Voltage Sensitivity for Atrial Sensing
Volt_V_Sensitivity	Voltage Sensitivity for Ventricular Sensing
V_Refractory_Period	Ventricular Refractory Period
A_Refractory_Period	Atrial Refractory Period
PVARP	Post Ventricular Atrial Refractory Period
Rate_Smoothing	Rate Smoothing
Activity_Threshold	Activity Threshold
Reaction_Time	Reaction Time
Response_Factor	Response Factor
Recovery_Time	Recovery Time
LRL_Period	Lower Rate Limit Period
AV_Delay	Atrioventricular Delay

Table 7: Serial input variable description

Each variable is first instantiated to a safe rate justified by the median value of acceptable values. The subsystem then waits for the end of a communication via COM port. These values are then saved as either 8-bit unsigned int, or a double precision floating point number as in the above figure.

Activity Level Analysis

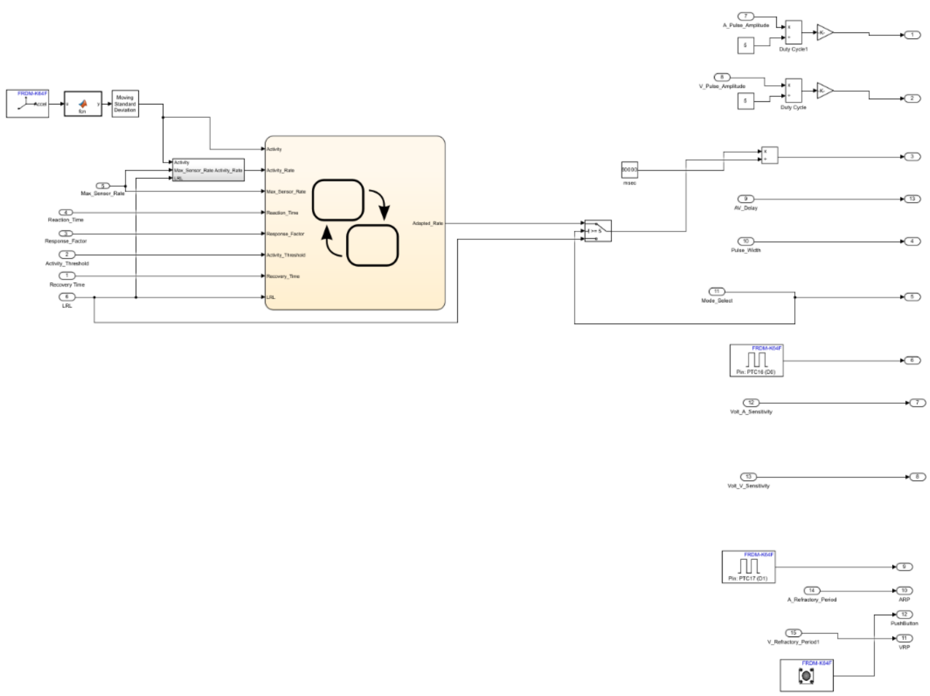


Figure 3: Activity Level Analysis Subsystem

The Activity Level Analysis consists of a subsystem with 15 from blocks. These blocks contain the variables received from the DCM. Once inside the Activity Level analysis subsystem the data is sent into another subsystem to compute the data. Before entering the Data Processing subsystem the accelerometer data is computed to be useful. Once this is done some of the incoming data in the Activity Level Analysis subsystem is send into the data processing subsystem to get a adapted rate value which is what decides the rate for the XXXR (Rate adaptive) modes.

Accelerometer

An accelerometer is a sensor on the board provided which reports back the acceleration of the sensor (or the device on which it is mounted) in the x, y and z direction. This data contains a lot of noise and sometimes is not useful in this form. Due to this our team decided that we had to normalize it in some fashion so that the data can be used. The first issue we cam across is that the data contains the acceleration in all three dimensions. When thinking about the purpose of the pacemaker we realized that the acceleration in all three dimensions is not useful. What our team cared about was more so the acceleration of the heart about all axis. Due to this we decided to take the magnitude of the incoming accelerometer data and combining all the data from the three different axis into one. This got rid of the three data sets but still contained much noise. To get rid of the noise our team decided to apply the moving standard deviation calculations on the incoming data. This allowed us to identify trends and changes in variability over time of the incoming data. This data was then run through another subsystem along with maximum sensor rate and LRL to get Activity Rate. This calculation was simply done to get the activity data from the accelerometer into a rate format. This data was then sent into the Data Processing subsystem mentioned before along with all the other data to be processed.

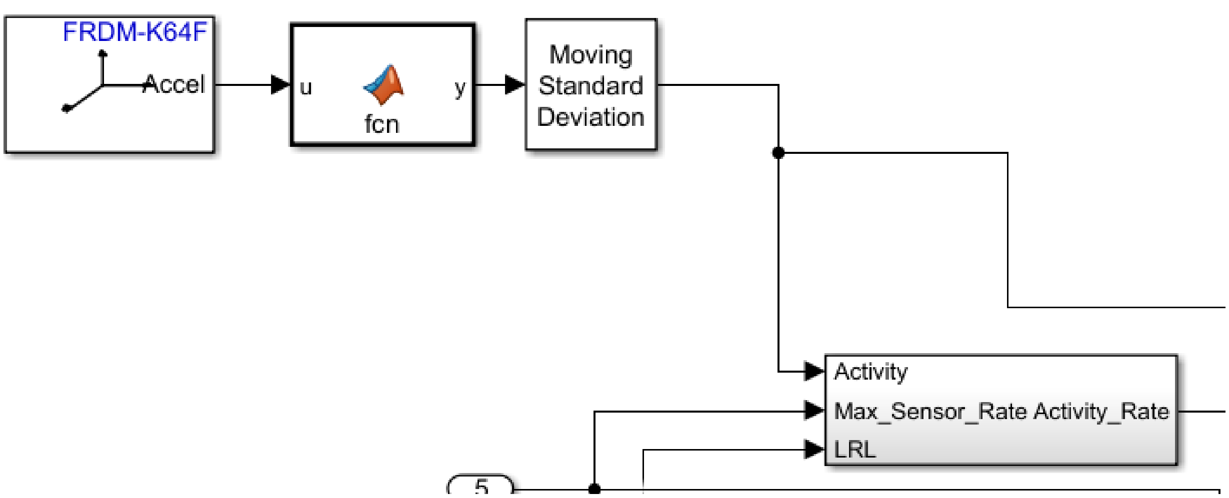


Figure 4: Accelerometer Logic

```
function y = fcn(u)
y = sqrt(u(1)^2+u(2)^2+u(3)^2);
```

Figure #5: Accelerometer Data Magnitude

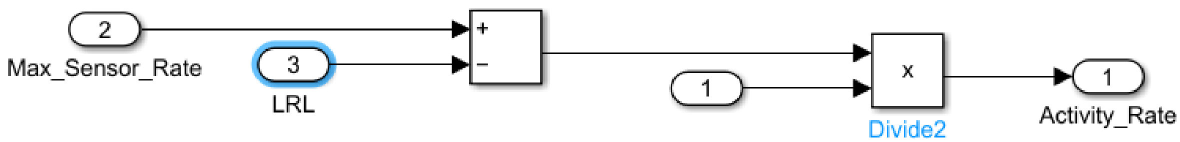


Figure 6: Activity Rate Logic

Data Processing Subsystem

The data processing Subsystem contains a state flow. The logic for this state flow is self-explanatory upon investigation except for two equations for the Adapted Rate variable. The adapted rate variable contains the rate at which to pulse the different chambers for the rate adaptive mode. These equations can be found in the states labelled “Up” and “Down”. For the “Down” state the adapted rate variable receives the variables Adapted_Rate, Rate_Diff, Recovery_Time and Time (time since entering the current state). Rate_Diff contains the difference between the Adapted Rate when this state was entered into and LRL. Using the Rate_Diff, Recovery_Time and Time variables a fractional component is determined which is then subtracted by the Adapted Rate to lower the Adapted Rate. In the “Up” state the variables being utilized are Response_Factor, Time, Reaction_Time Activity Rate and LRL. For this section The LRL value is taken as default and to it a fraction multiplied by the Response factor and the activity rate is added. Important point to note is that this fraction is always increasing when in the “Up” state. To satisfy the requirement of the rate not increasing past the maximum sensor rate and not dropping the decisions to change states are mentioned as they are. A general flow of this stateflow is the following. The Adapted Rate is set to LRL. The operating system moves to the MIN state while constantly checking the Activity and comparing it to the Activity_Threshold if it exceeds or is equal to Activity the operating system moves to the Up state. If the activity decreases past the threshold then it moves to the DOWN state, but if the Adapted rate increases past the Max sensor rate then it enters the MAX state where the Adapted rate is fixed to Max Sensor rate. From here if the activity decreases past the threshold it goes to the DOWN state. If in the DOWN state activity the activity exceeds the threshold then it is sent back into the UP state.

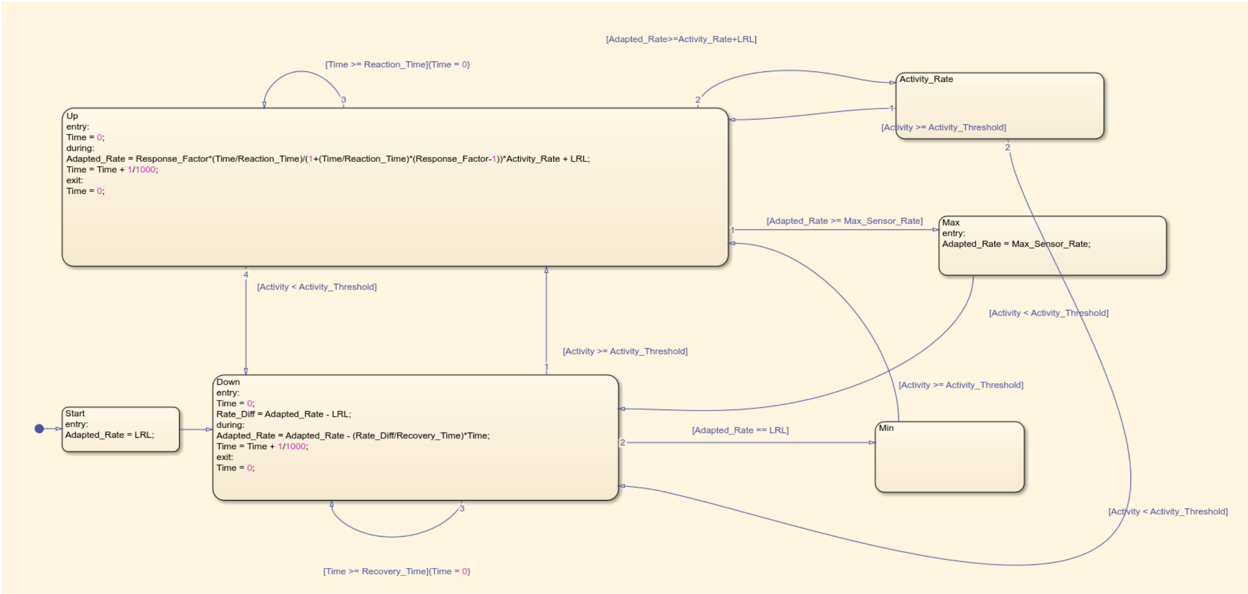


Figure 7: Data Processing Subsystem

Main State Machine

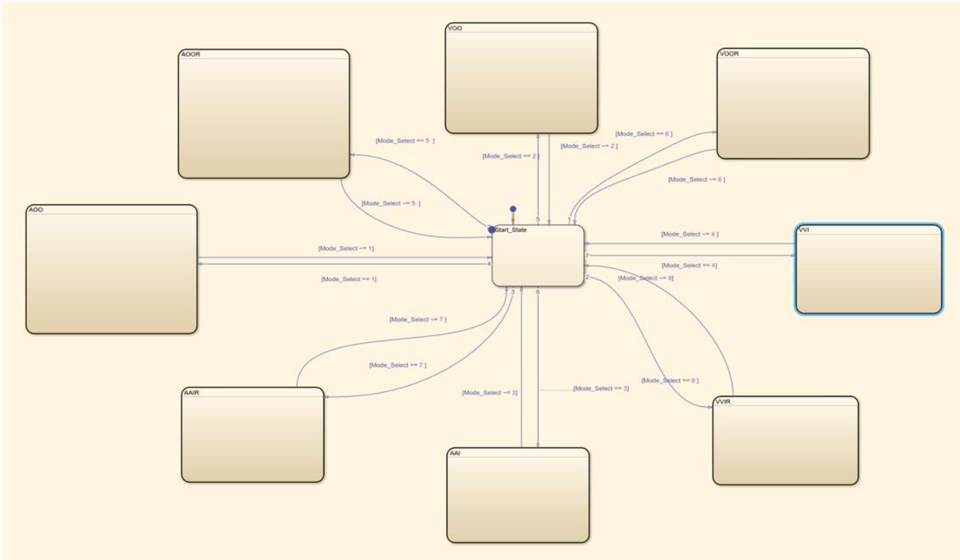


Figure 8: Main State Machine Subsystem

The Main State Machine module contains all pacing calculations and mode selections. Based on the Mode_Select variable, modes VVI, VVIR, AAI, AAIR, AOO, AOOR, VOO, VOOR are selected from. Within each susbsystem, ventrical or atrium capicators are charged with VENT/ATR_PACE_CTRL VENT/ATR_GND_CTRL and PACE_GND_CTRL. After sensing of either ventricle or atrium pulses respective of pacing mode, the capicator is discharged with the opposite signal sent to VENT/ATR_PACE_CTRL VENT/ATR_GND_CTRL and PACE_GND_CTRL.

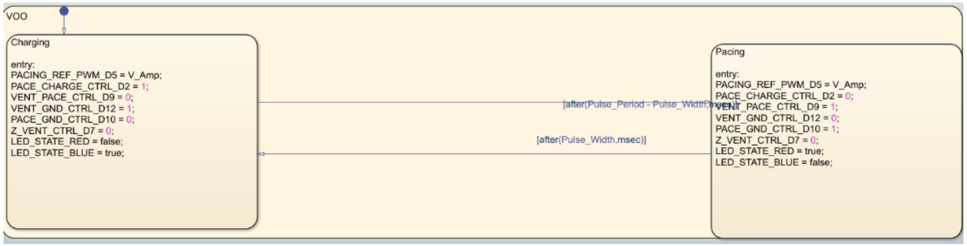


Figure 9: VOO Pacing Logic

Figure 10: VOOR Pacing Logic

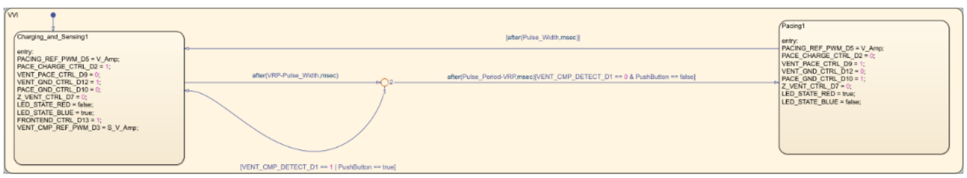


Figure 11: VVI Pacing Logic

Figure 12: VVI Pacing Logic

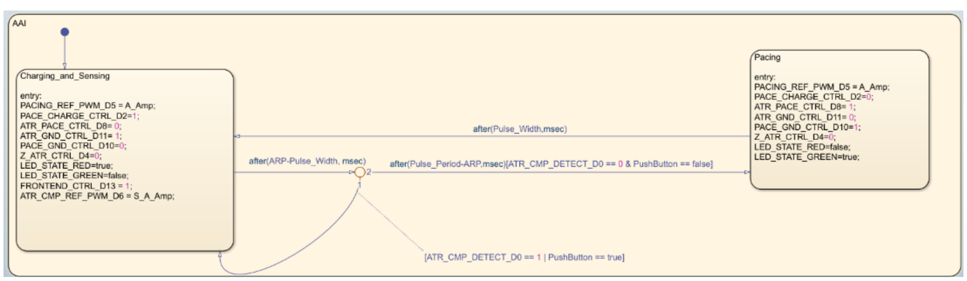


Figure 13: AAI Pacing Logic

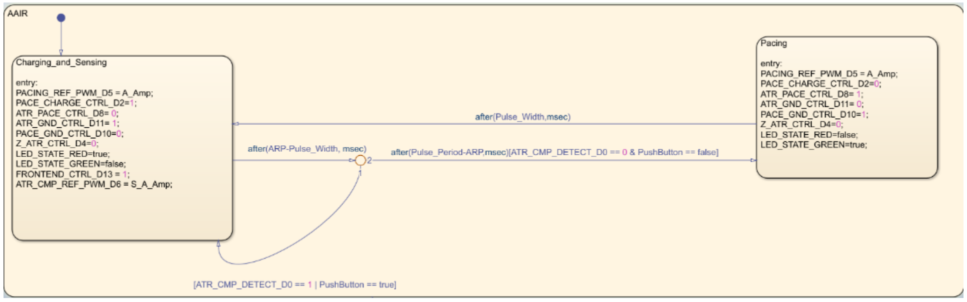


Figure 14: AAIR Pacing Logic

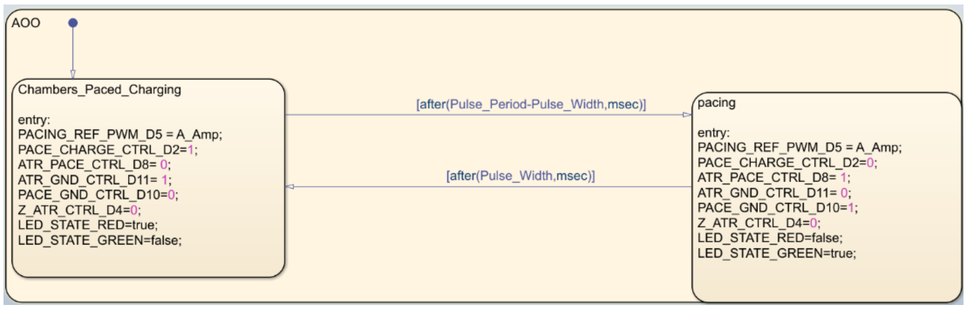


Figure 15: AOO Pacing Logic

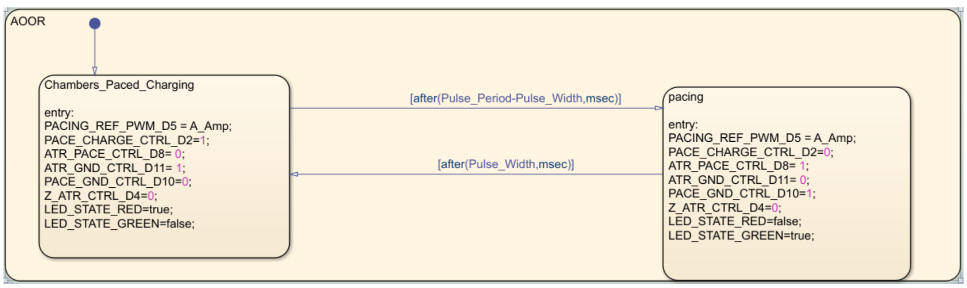


Figure 16: AOOR Pacing Logic

Hardware Pin Output

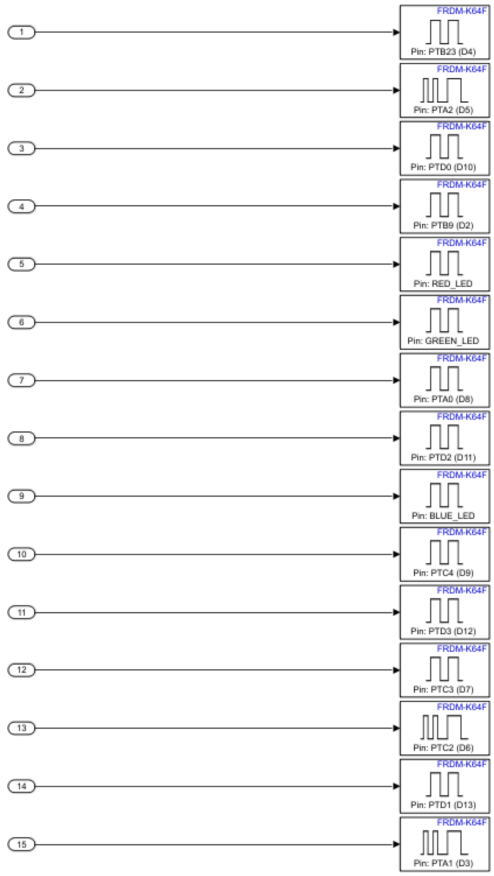
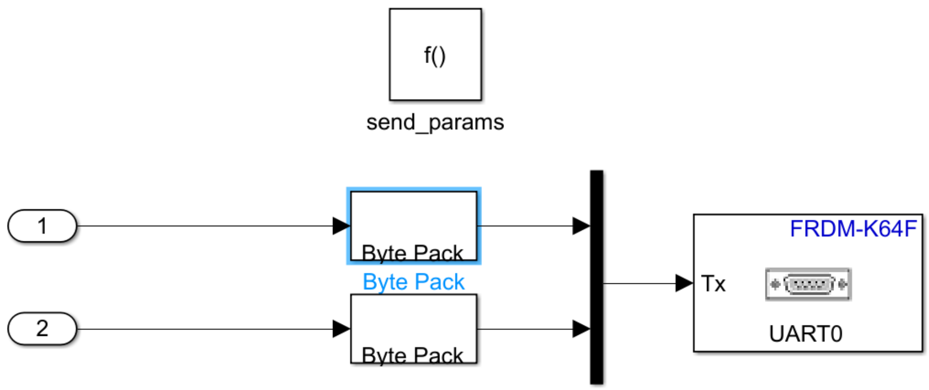


Figure 17: Hardware Pin

The Hardware Pin Output is the simplest step. All calculations and saved values within inports are sent to its corresponding pins on the board. The mapping can be seen in the figure above.

EGRAM Data Output

EGRAM data is always continuously sent to the DCM via COM port. This data includes the atrium signal (ATR_SIGNAL) given by PTB2 (A0) and ventricle signal (VENT_SIGNAL) given by PTB3(A1). These are first stored on inports before being converted to bytepacks with type double and sent to the COM port via MUX.



Device Control-Monitor

Below you will find the DCM module description covering in detail for each page and functions of code blocks.

DCM Module Description

Classes and methods in function library:

class NoUserError, UserExistError, PasswdMatchError, NoDatabaseError, UnpicklingError, InputNotValidError

Inherited from *Exception*. Containing the string descriptions of each error.

def hash

Parameter: data.

Return Value: SHA256 of *data*.

def center_window

Centers the input window instance.

Parameter: win: tkinter.Tk.

def list_serial_ports

Return Value: List of str representing all serial ports connected to computer.

class VerticalScrolledFrame

Inherited from *tkinter.ttk.Frame*. Creates a frame with scroll bar and mouse wheel support.

class LoadSerialEgram

Inherited from *threading.Thread*. Creates a thread that reads Egram data from selected serial and call *DrawEgram* to draw from read data.

class DrawEgram

Read data in *LoadSerialEgram* every selected time period and draw with *matplotlib*.

Front-end related classes:

class AskString

Description: creates a *tkinter.Toplevel* popup to ask a string from user.

class login

Description: responsible for drawing login page and handling login page data.

Instance variables:

root: tkinter.ttk.Tk

userid: int

login_success: bool

login_name: str

Class methods:

def __new__

Constructor. Creates/initializes tkinter window instance and initialize window components and variables.

Parameters:

relogin: bool(False). Controls whether a new tkinter window is needed. Creates a new tkinter window by default.

quit_code: int|str(1). The reason the previous session is closed. 1 indicates the user logout; any string indicates the user is changing their password, the string is the username.

Return value: tuple(root, userid, login_success, login_name)

root: Root page. Passing to the following classes to avoid bugs.

userid: Login user’s id. -1 if not logged in.

login_success: True if the user successfully logged in.

login_name: User’s default name. Passing to the following classes.

def login

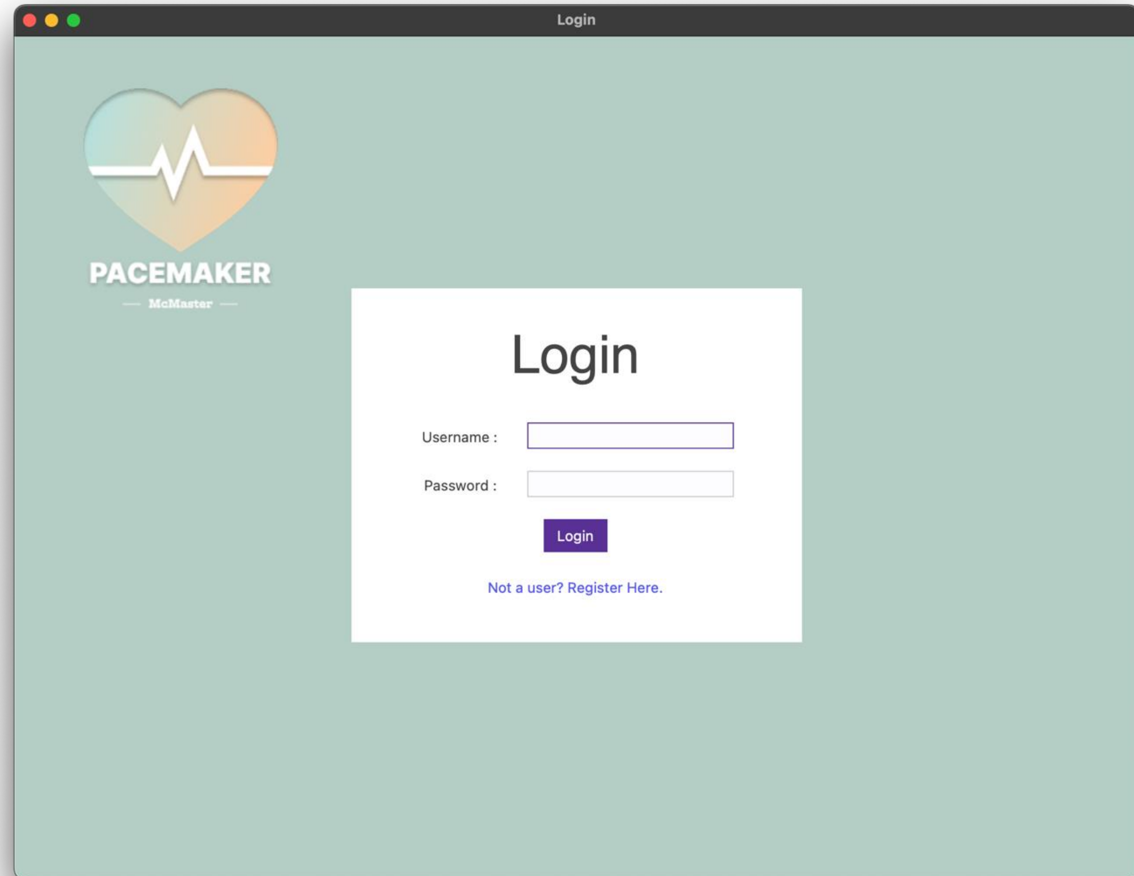


Figure 19: Account Login Page

To draw the login form.

Parameters:

username: str(“”). The default username to be displayed in the login form. Passed by registration or change password page. Leave blank by default.

```
def register
```

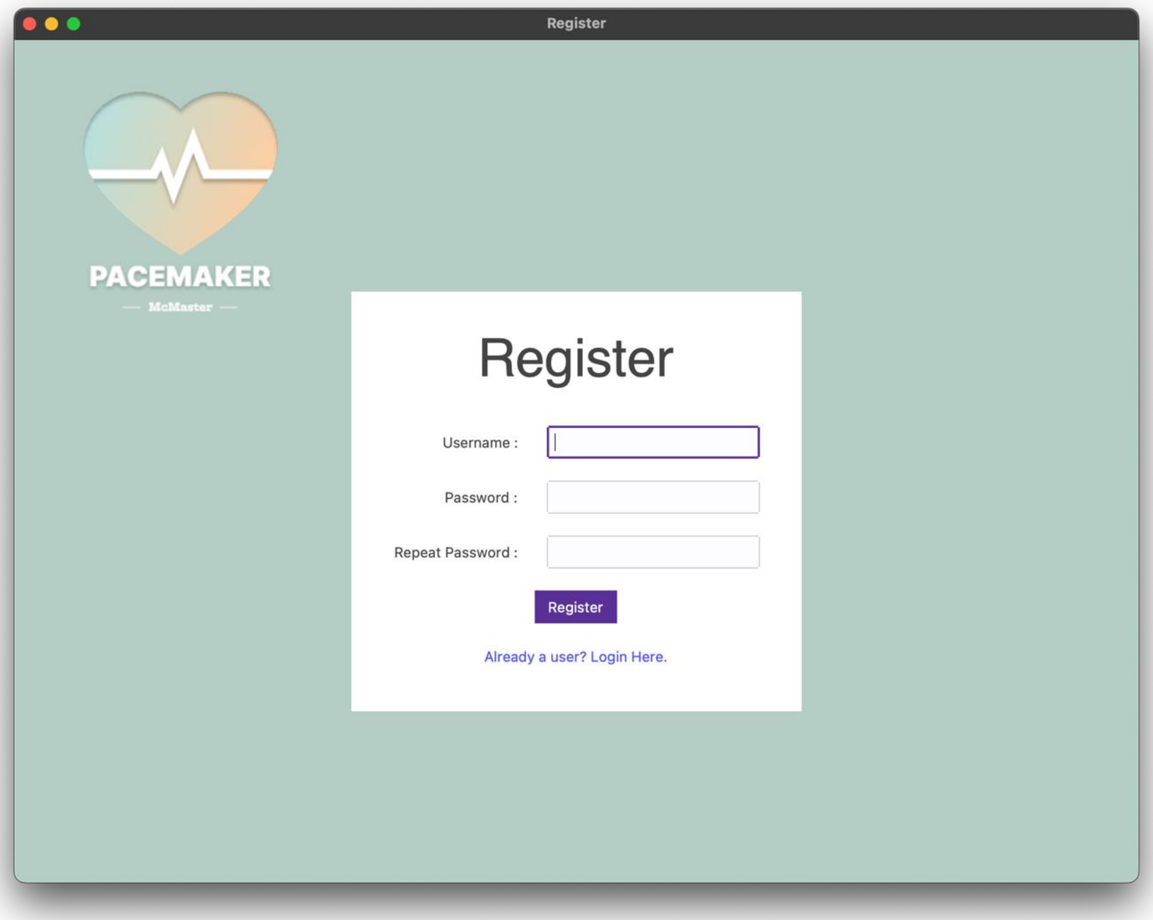


Figure 20: Account Registration Page

To draw the registration form.

Parameters:

username: str(“”). The default username to be displayed in the login form. Passed by login page. Leave blank by default.

```
def change_password
```

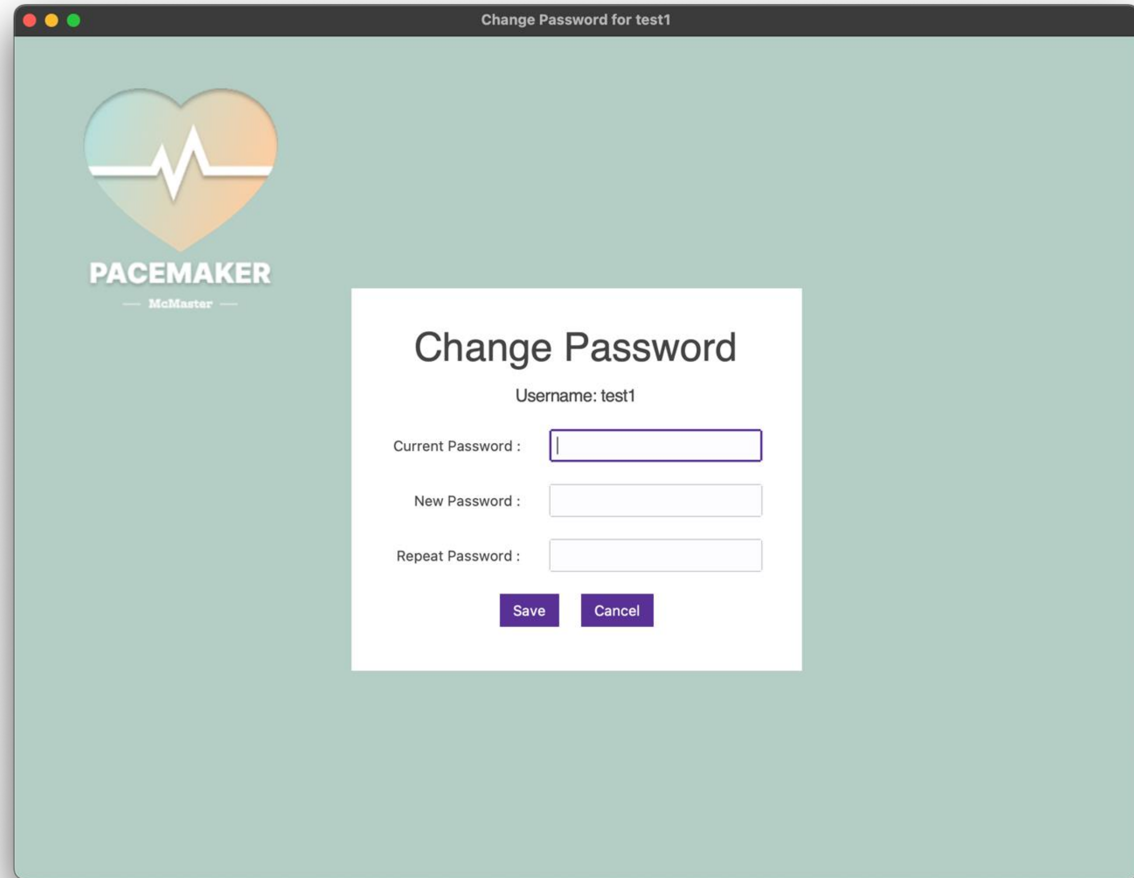


Figure 21: Account Password Reset Page

To draw the change password form.

Parameters:

username: str. The username for which the password is to be changed.

def validate_login

Called by *login*. Back-end method that validates user credentials.

Parameters:

user_e: tk.Entry. Entry widget containing the username.

passed_e: tk.Entry. Entry widget containing the password.

Automatically call *register* if user doesn't exist through *error_handler*.

def create_user

Called by *register*. Back-end method that creates new account.

Parameters:

user_e: tk.Entry. Entry widget containing the username.

passed_e: tk.Entry. Entry widget containing the password.

re_passed_e: tk.Entry. Entry widget containing the repeated password.

Automatically call *login* if user already exist through *error_handler* or successfully created.

def validate_newpass

Called by *register*. Back-end method that creates new account.

Parameters:

username: str. The username for which the password is to be changed.

old_e: tk.Entry. Entry widget containing the old password.

passed_e: tk.Entry. Entry widget containing the password.

re_passed_e: tk.Entry. Entry widget containing the repeated password.

cancel: bool(False). Cancels the password change process and automatically re-login into *username* if True.

Automatically call *login* if password successfully changed.

def error_handler

Handles errors raised during login and registration processes, including *NoUserError*, *UserExistError*, *PasswdMatchError*, *NoDatabaseError*, *UnpicklingError*, *InputNotValidError*.

Parameters:

err: Exception. The exception that was raised.

username: str. the username associated with the error.

Calls *login*, *register* respect to the error if needed.

class mainpage

Description: responsible for drawing main page and handling user data.

Global variables:

root: Root page. From *login*.

userid: Login user's id. From *login*.

login_success: From *login*.

login_name: User's default name. From *login*.

zoomfactor: Base on platforms. 1 on Darwin/Linux and 1.8 for Windows to add HiDPi support for Windows.

Instance variables:

lang: langpack.LangDict(EN). Language pack stored in langpack. English by default.

lang_var_str: str("EN"). String representing language selected. English by default.

serial_status: bool(False).

pacing_mode: str.

rootw: int(800*zoomfactor). Root window minimum width.

rooth: int(560*zoomfactor). Root window minimum height.

presets: dict. Dictionary stores current user's presets.

showing_egram: bool. A flag for wheather showing the Egram.

t: LoadSerialEgram. A thread that would automatically load Egam data from serial.

quit_code: int(0).

mode: str("MAIN").

serial_port: str.

Class methods:

`def init_self_pref`

Load user preferences from file or create the database if database does not exist.

`def sync_self_pref`

Sync user preferences to file.

`def quit`

Finalize data base on different reasons of closing.

Parameters:

mode: str("EXIT").

`def __new__`

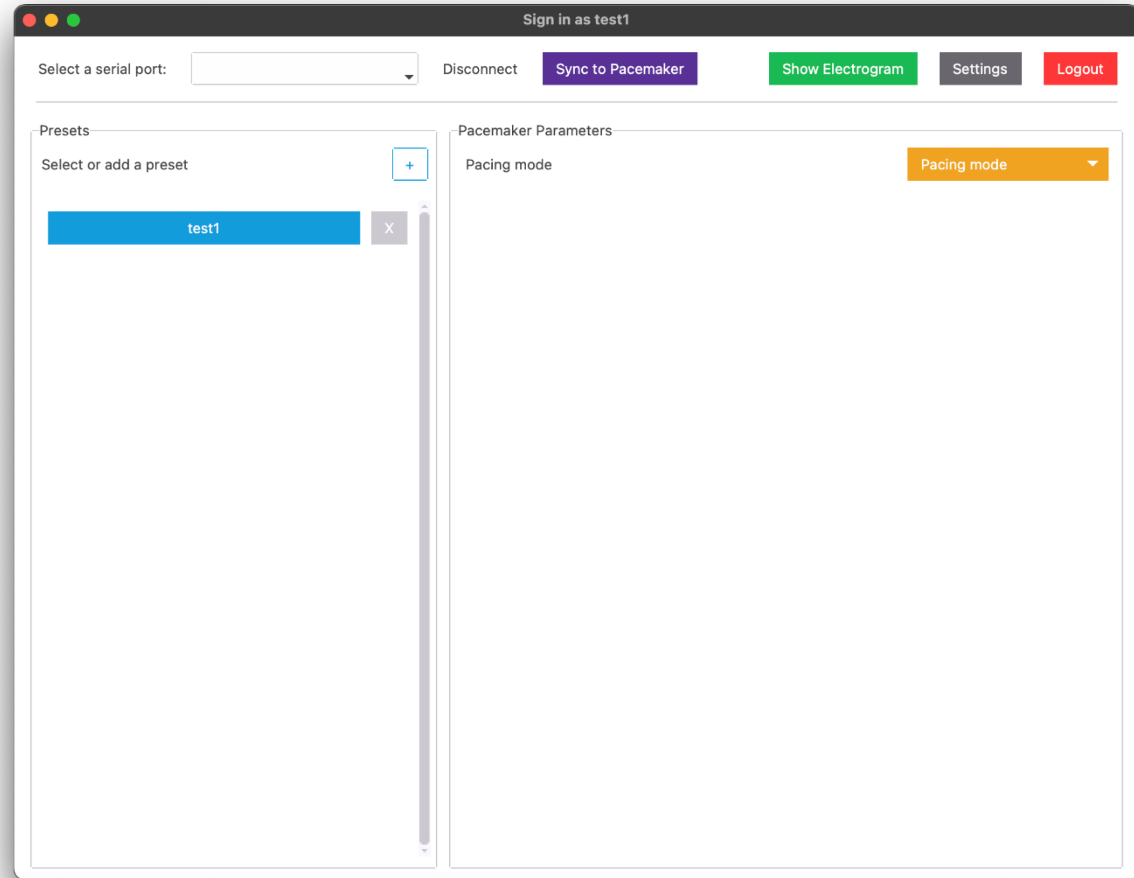


Figure 22: Pacing Mode Select Screen

Constructor. Creates/initializes tkinter window instance and initialize window components and variables.

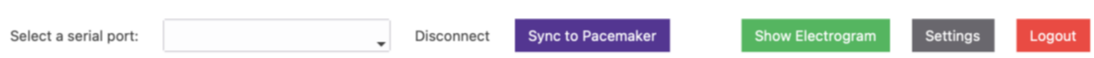
Parameters:

userid: int

login_name: str

root: tkinter.ttk.Tk

def toolbar_frame



Draws/updates toolbar base on the status *stat*. Draws Main toolbar by default.

Parameters:

stat: str("Main").

`def pacemaker_configure_frame`

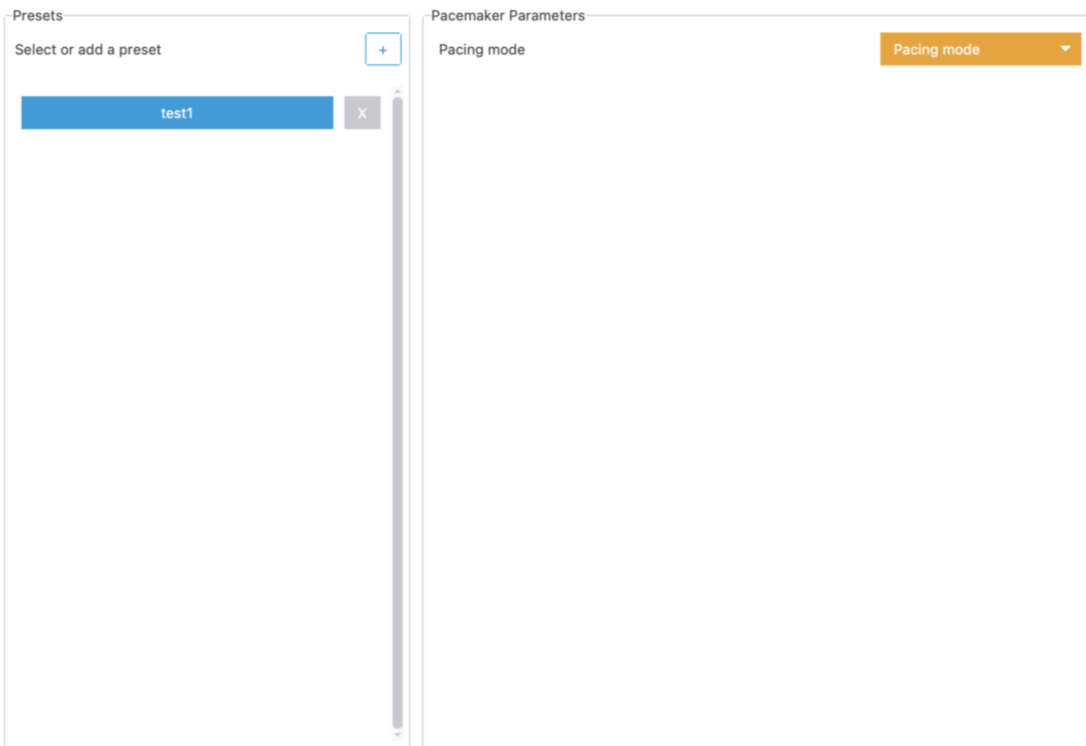


Figure 23: Preset Selection Screen

Figure 24: EGRAM preview and Parameter Selection Screen

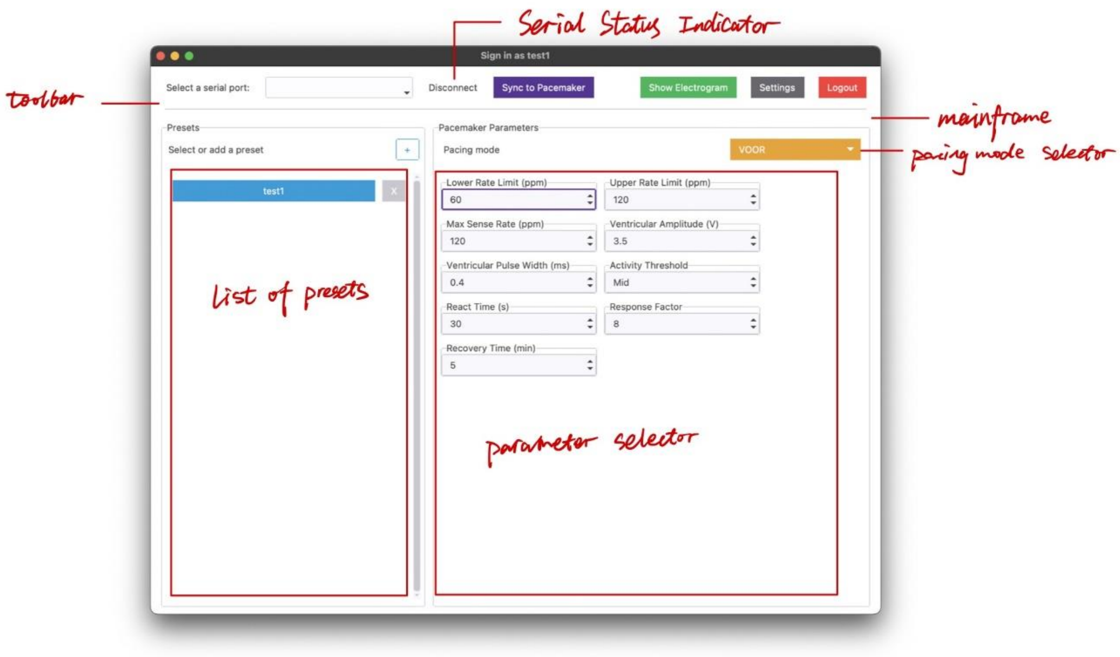


Figure 25: Initial Parameter Selection Screen with Labeled elements

Draws/updates pacemaker frame base on the status *stat*. Initializes preset frame and parameter frame.

```
def update_pacing_para_frame
```

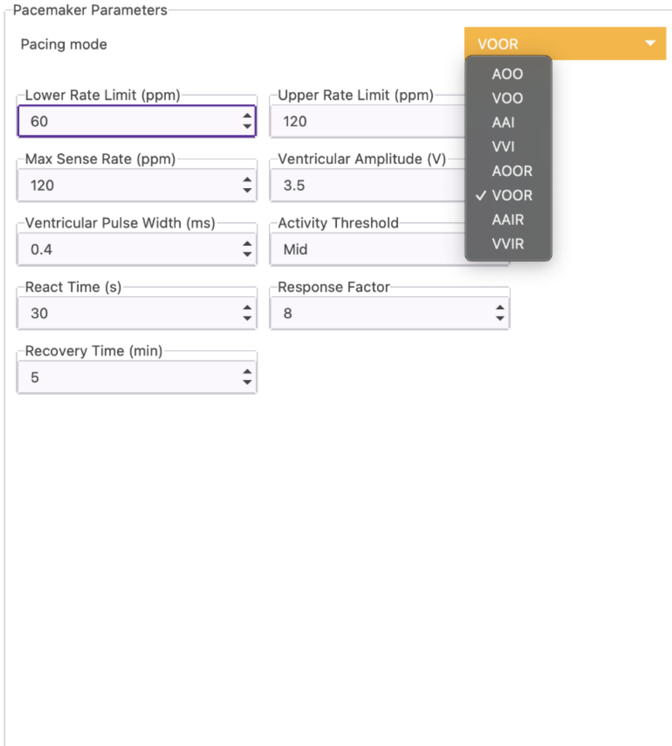


Figure 26: VOOR Parameter Selection Screen

Draws pacemaker parameter frame or loads presets.

Parameters:

load_preset: str. The name of preset to be loaded. Null by default.

def get_current_paras

Get current parameters set in parameter frame.

Return values: List of str. Containing each parameter as string.

def add_preset

Adds preset to *self.presets*. Creates a AskString instance to ask the user for preset name.

def load_preset

Loads preset from *self.presets*.

Parameters:

preset_name: str. The name of preset to be loaded.

if_ask: bool(True). Confirm loading if True.

def del_preset

Deletes preset from *self.presets*.

Parameters:

preset_name: str. The name of preset to be deleted.

def update_preset_frame

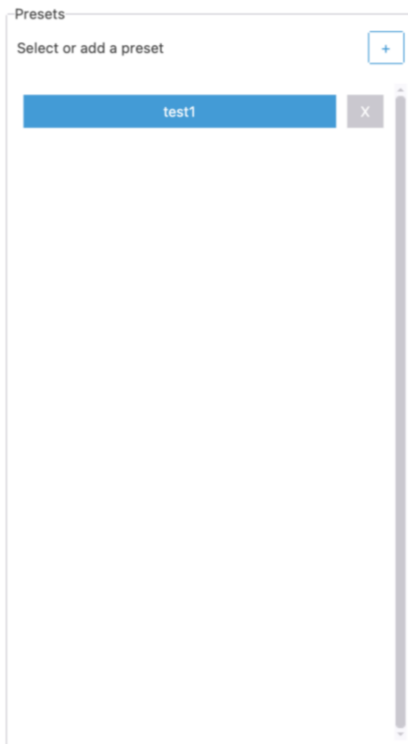


Figure 27: Preset Save Panel

Updates preset frame.

def show_egram

Draws and initializes Egram frame. Starts *self.t* to read Egram data from serial in the background.

def hide_egram

Destroys Egram frame. Stops *self.t*.

def setting_frame

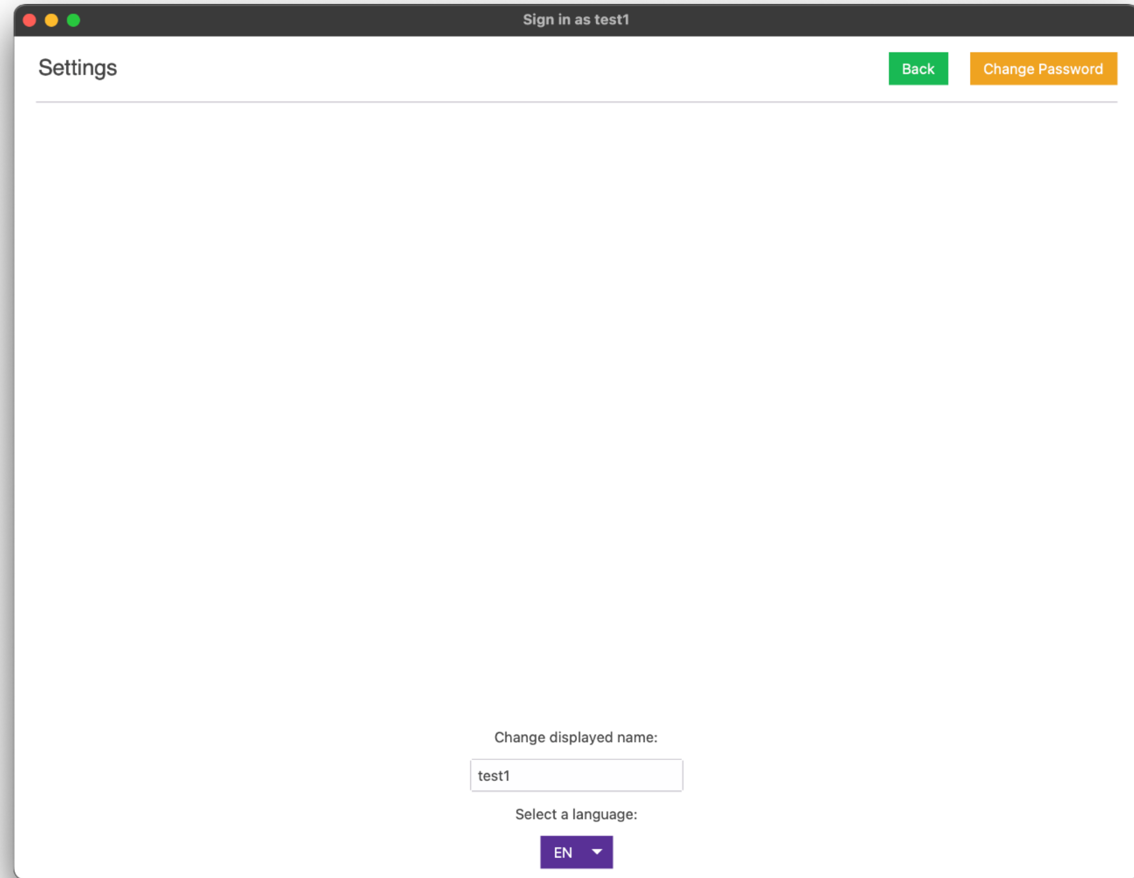


Figure 28: Language Selection and Name Change Screen

Draws settings frame.

Parameters:

reload: bool(False). Only true while changing the language.

def change_display_name

Private call back function. Updates the display name in status bar set by user.

Parameters:

display_name_var: tkinter.StringVar. Contains the new login name string.

def set_lang

Private call back function. Updates the entire window with the language the user selected.

DCM Features

Multi-language support:

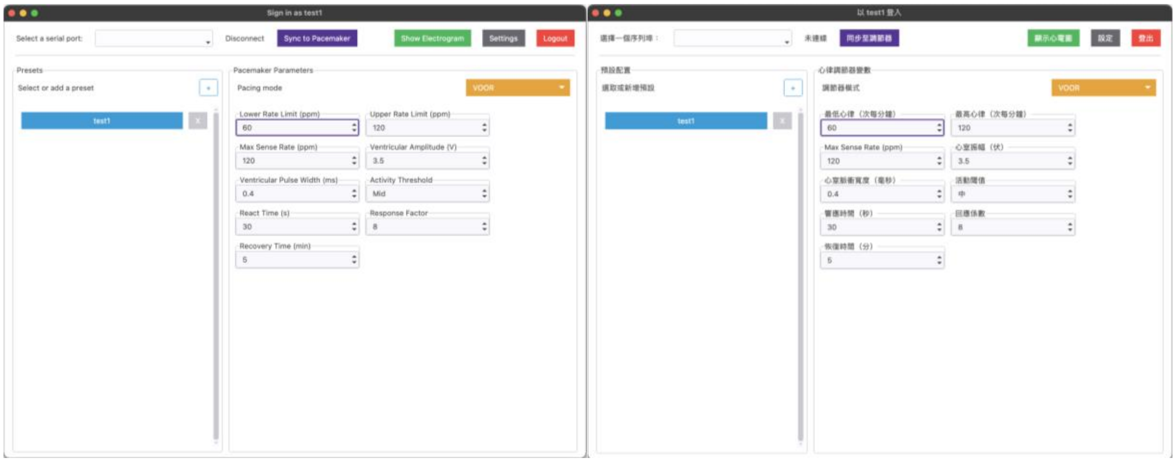


Figure 29: English and Chinese Language Support on Parameter Selection Screen

Support English and Traditional Chinese. Can be changed in settings. Autosaved for each user. Language support are in separate file for convenience to add other languages.

User-friendly Login:

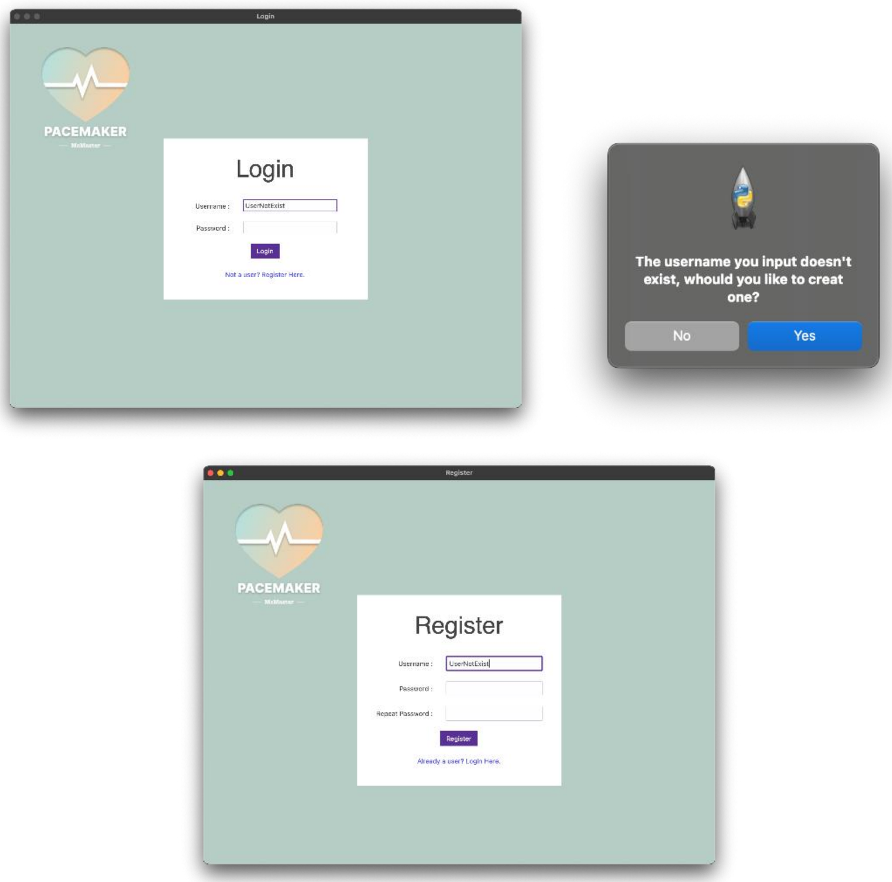


Figure 30: Login & Registration Screens with User Feedback on Registration Attempt

Would automatically pass the username to registration if the user does not exist. Similarly

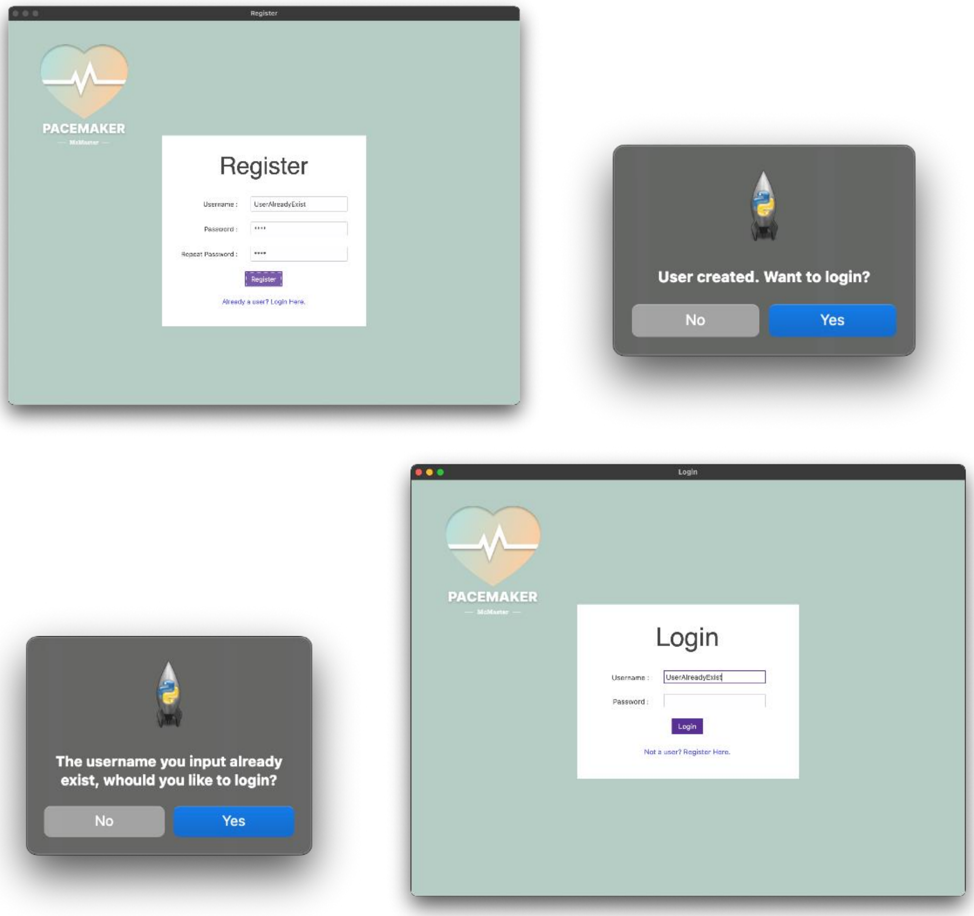


Figure 31: Login & Registration Screens with User Feedback on User Creation and Valid Username Input

User-friendly Main Page:

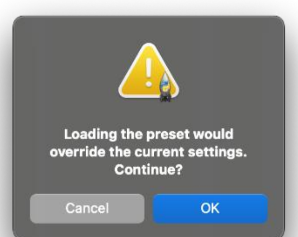


Figure 32: Loading Preset Warning

Change the user's data if and only if with consent.

After re-login after language and parameters changed:

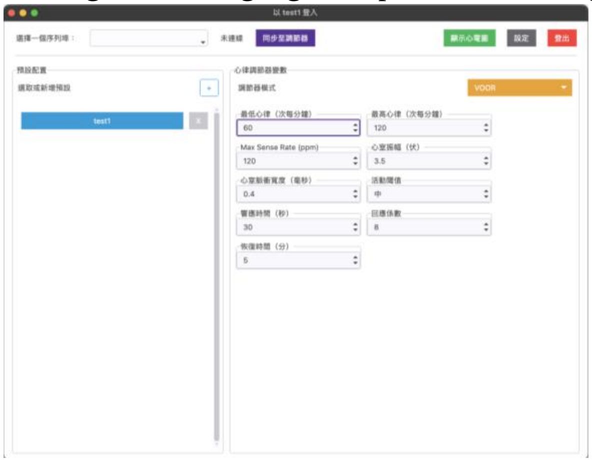


Figure 33: Saved Values After DCM Relogin

Language, display name, current parameters, presets are automatically saved. The interface would remain exactly the same while user re-login.

Testing

Pacemaker (Simulink)

AOO and VOO

To ensure the accurate verification of these pacing modes, two tests need to be conducted. The initial test involves allowing the pacemaker to operate without any inherent heartbeats, while the subsequent test involves introducing heartbeats to examine if the pacemaker functions irrespective of the natural beats. The results produced by these pacing modes are identical, differing only in the chamber used. The test scenarios provided confirm AOO but can also be employed and compared with VOO outputs.

Pacemaker Settings		Heartview Settings	
Mode	AOO	Natural Atrium	Off
Pulse Width (ms)	10	Natural Ventricle	Off
Pulse Amplitude (V)	4	Natural Heart Rate (BPM)	30
Lower Rate (PPM)	10	Natural AV Delay (ms)	30

Output	
Expected	Actual
The pacemaker continuously paces the heart with an interval of the Lower Rate.	The result obtained was the expected output.
Test result - *Passed*	
Change required – *N/A*	

Table 9: AOO & VOO Test Case #1

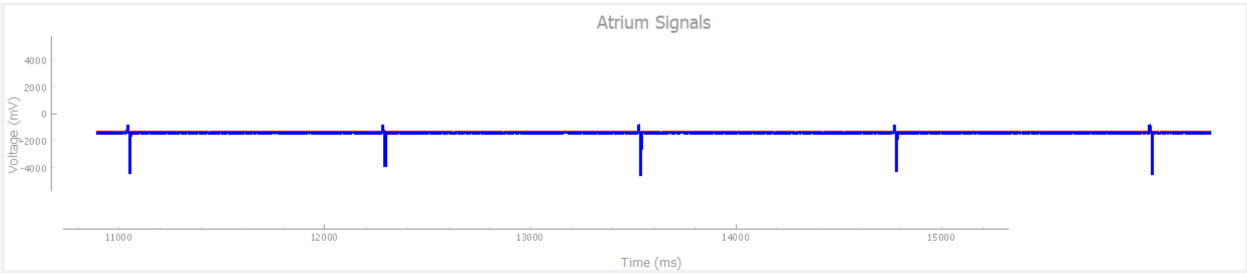


Figure 33: Atrium signal on HeartView™ for AOO/VOO test case #1.

Input			
Pacemaker Settings		Heartview Settings	
Mode	AOO	Natural Atrium	10 ms
Pulse Width (ms)	10	Natural Ventricle	10 ms
Pulse Amplitude (V)	4	Natural Heart Rate (BPM)	60
Lower Rate (PPM)	10	Natural AV Delay (ms)	30
Output			
Expected		Actual	
The pacemaker continuously paces the heart with an interval of the Lower Rate.		The result obtained was the expected output.	
Test result - *Passed*			
Change required – *N/A*			

Table 10: AOO & VOO Test Case #2

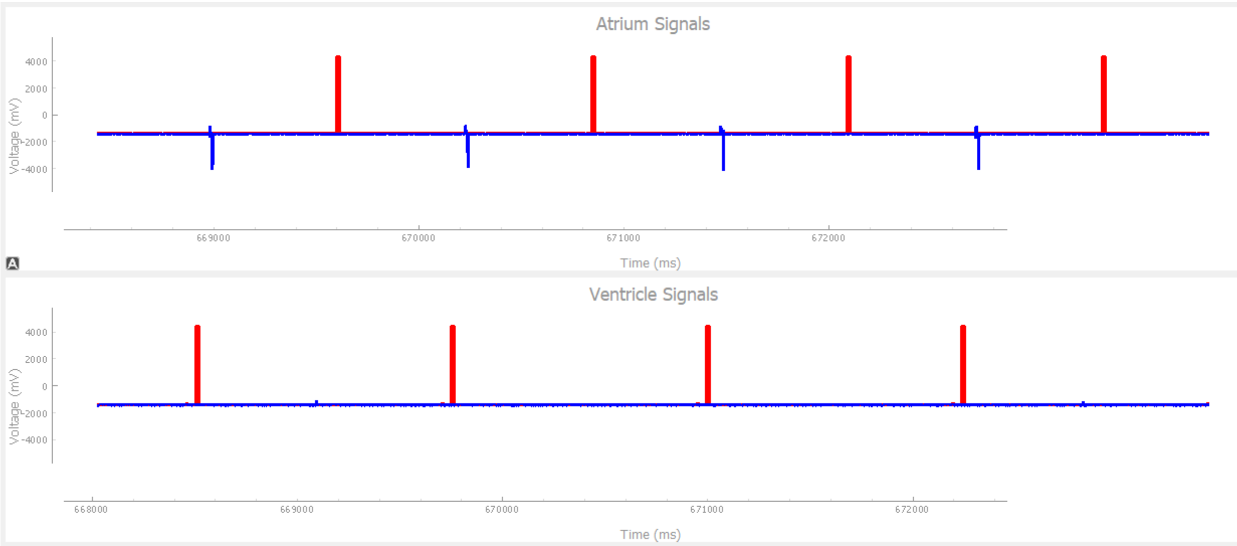


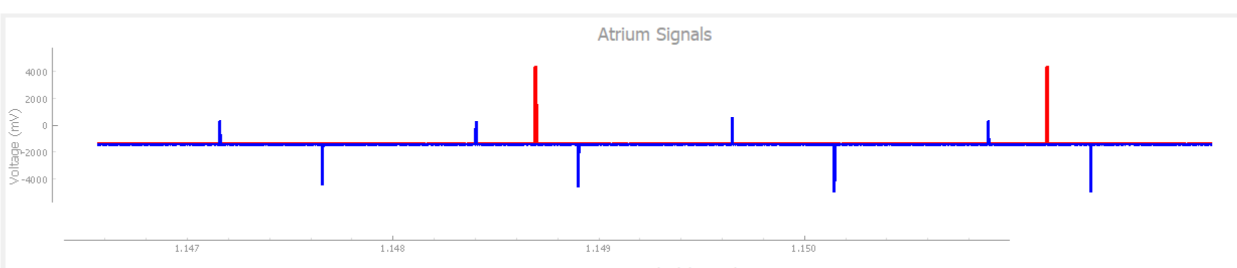
Figure 34: Atrium signal on HeartView™ for AOO/VOO test case #2.

AAI and VVI

The test cases for both AAI and VVI exhibit similarities in their logic, resulting in comparable outcomes. There will be a total of four cases examined. While this doesn't cover every conceivable scenario, it aims to test fundamental and extreme conditions.

AAI and VVI: Test Case #1

Input			
Pacemaker Settings		Heartview Settings	
Mode	AAI	Natural Atrium	5 ms
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	30
Lower Rate (PPM)	100	Natural AV Delay (ms)	30
Output			
Expected		Actual	



The pacemaker paces the heart only when the heart cannot adequately pace itself. It initiates pacing activities between the heartbeats and pauses these paces when the heart naturally beats.	The result obtained was the expected output.
Test result - *Passed*	
Change required – *N/A*	

Figure: Atrium signal on HeartView™ for AAI/VVI test case #1

AAI and VVI: Test Case #2

Input			
Pacemaker Settings		Heartview Settings	
Mode	AAI	Natural Atrium	10 ms
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
The pacemaker paces the heart only when the heart cannot adequately pace itself. It should not pulse the heart at all as it is pulsing itself naturally.		The result obtained was the expected output.	
Test result - *Passed*			
Change required – *N/A*			

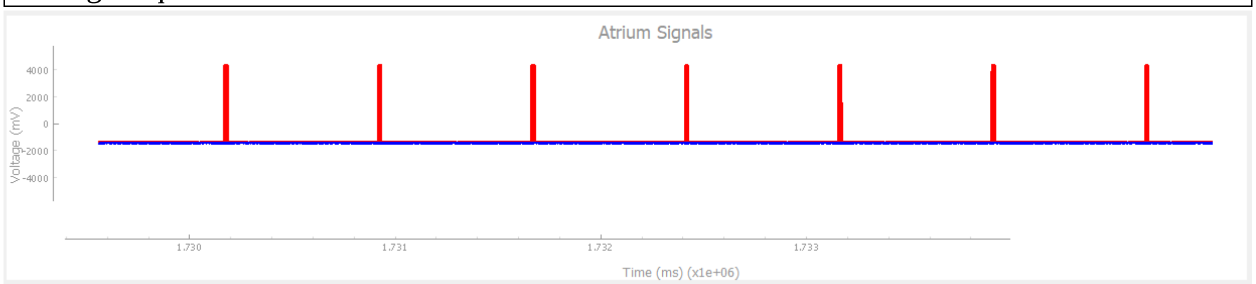


Figure 35: Atrium signal on HeartView™ for AAI/VVI test case #2

AAI and VVI: Test Case #3

Input			
Pacemaker Settings		Heartview Settings	
Mode	AAI	Natural Atrium	Off
Pulse Width (ms)	400	Natural Ventricle	Off

Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
The pacemaker paces the heart only when the heart cannot adequately pace itself. It should pulse the heart continuously as the heart is not pulsing itself naturally.		The result obtained was the expected output.	
Test result - *Passed*			
Change required – *N/A*			

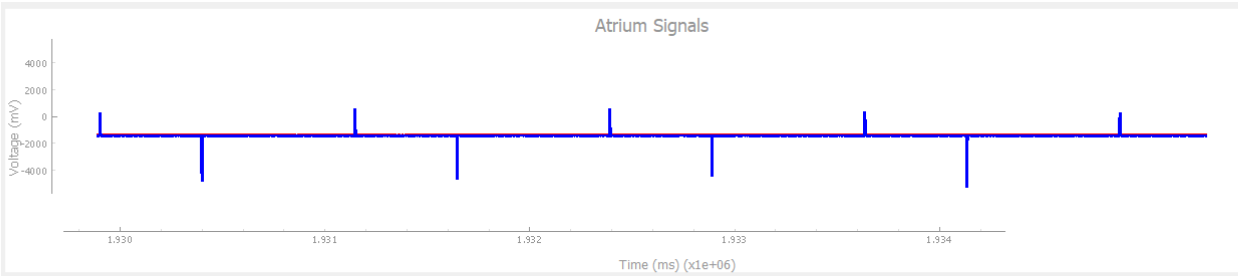


Figure 36: Atrium signal on HeartView™ for AAI/VVI test case #3

AAI and VVI: Test Case #4

Input			
Pacemaker Settings		Heartview Settings	
Mode	AAI	Natural Atrium	10 ms
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	98
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
The pacemaker paces the heart only when the heart cannot adequately pace itself. It should inhibit all the pulses that are outside ARP.		The result obtained was the expected output.	
Test result - *Passed*			
Change required – *N/A*			

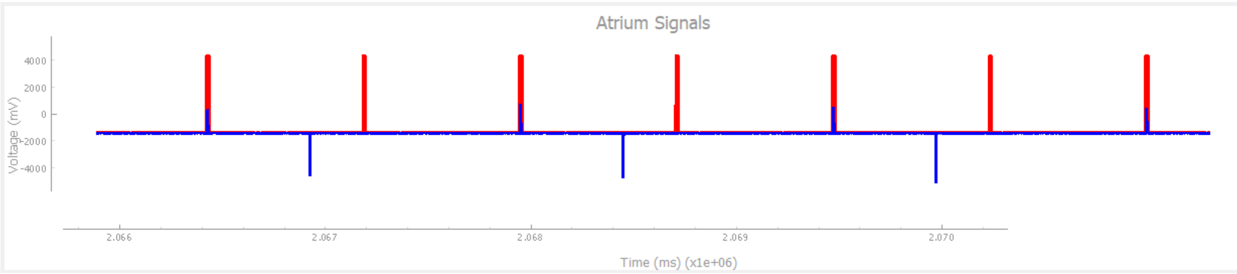


Figure 37: Atrium signal on HeartView™ for AAI/VVI test case #4

AOOR and VOOR

To accurately validate the adaptive pacing modes, two tests need to be conducted. The initial test involves introducing activity to monitor whether the pacing rate gradually escalates. The second test involves introducing natural heartbeats during activity and observing the pacing changes while shaking the board. The results generated by these pacing modes remain identical, differing only in the chamber used.

AOOR and VOOR: Test Case #1

Input			
Pacemaker Settings		Heartview Settings	
Mode	AOOR	Natural Atrium	Off
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
When the accelerometer is shaken, the pacemaker will elevate its pacing rate.		The result obtained shows that the atrium was not paced at all.	
Test result - *Failed*			
Change required – *The atrium should be paced.*			

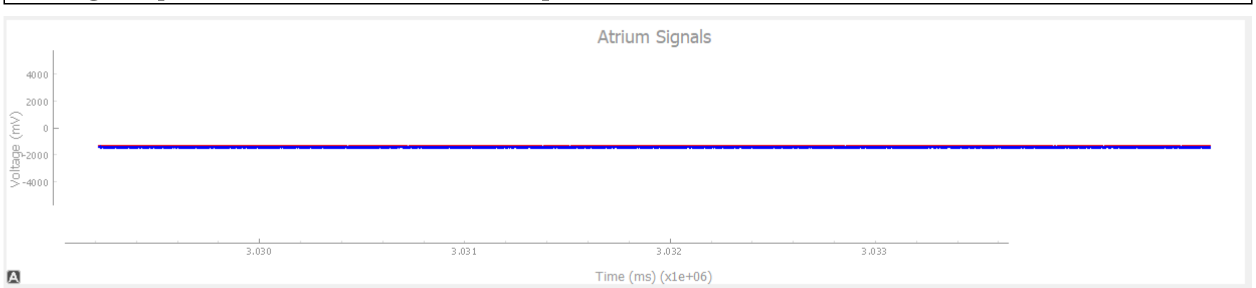


Figure 38: Atrium signal on HeartView™ for AOOR/VOOR test case #1

AOOR and VOOR: Test Case #2

Input			
Pacemaker Settings		Heartview Settings	
Mode	AOOR	Natural Atrium	10 ms
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
When the accelerometer is shaken, the pacemaker will elevate its pacing rate.		The result obtained shows that the atrium was not paced at all.	
Test result - *Failed*			
Change required – *The atrium should be paced.*			

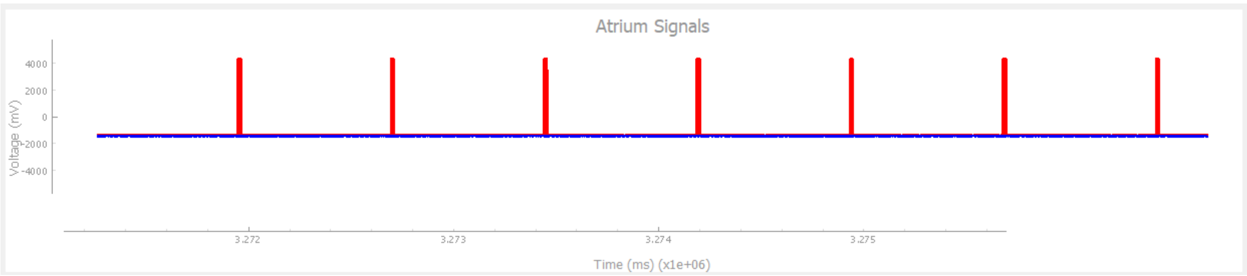


Figure 39: Atrium signal on HeartView™ for AOOR/VOOR test case #2

AOOR and VOOR: Test Case #3

Input			
Pacemaker Settings		Heartview Settings	
Mode	VOOR	Natural Atrium	Off
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
When the accelerometer is shaken, the pacemaker will elevate its pacing rate.		The result obtained shows that the ventricle was not paced at all.	
Test result - *Failed*			
Change required – *The* ventricle *should be paced.*			

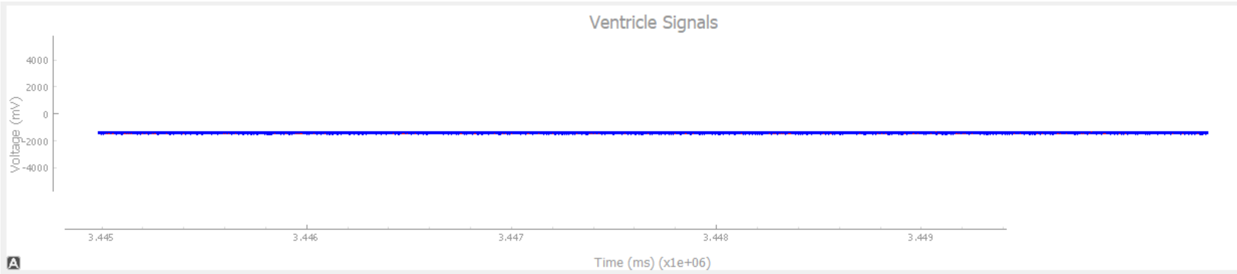


Figure 40: Venticle signal on HeartView™ for AOOR/VOOR test case #3

AOOR and VOOR: Test Case #4

Input			
Pacemaker Settings		Heartview Settings	
Mode	VOOR	Natural Atrium	Off
Pulse Width (ms)	400	Natural Ventricle	10 ms
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
When the accelerometer is shaken, the pacemaker will elevate its pacing rate.		The result obtained shows that the ventricle was not paced at all.	
Test result - *Failed*			
Change required – *The* ventricle *should be paced.*			

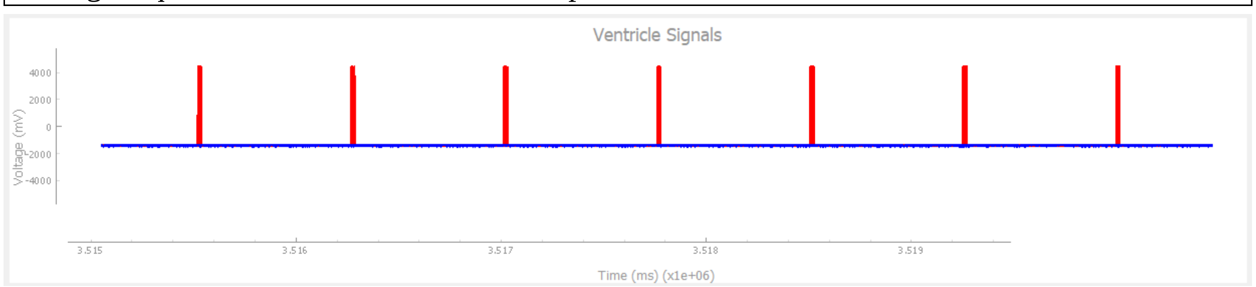


Figure 41: Venticle signal on HeartView™ for AOOR/VOOR test case #4

AAIR and VVIR

To validate these modes, two tests are required. The initial test aims to determine whether the pacemaker will refrain from pacing if the Lower Rate matches the Natural Heart Rate; however, upon shaking the board, pacing should occur. The second test involves setting the pacemaker's Lower Rate higher than the Natural Heart Rate to ascertain whether it will suppress pacing. Subsequently, shaking the pacemaker should result in more pacing instances.

AAIR and VVIR: Test Case #1

Input			
Pacemaker Settings		Heartview Settings	
Mode	AAIR	Natural Atrium	10 MS
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
When the Natural Heart Rate matches the Lower Rate, the pacemaker should not suppress pacing. Pacing should be observed when the pacemaker is shaken.		The result obtained shows that the atrium was not paced when it was shaken.	
Test result - *Failed*			
Change required – *The atrium should be paced when shaken.*			

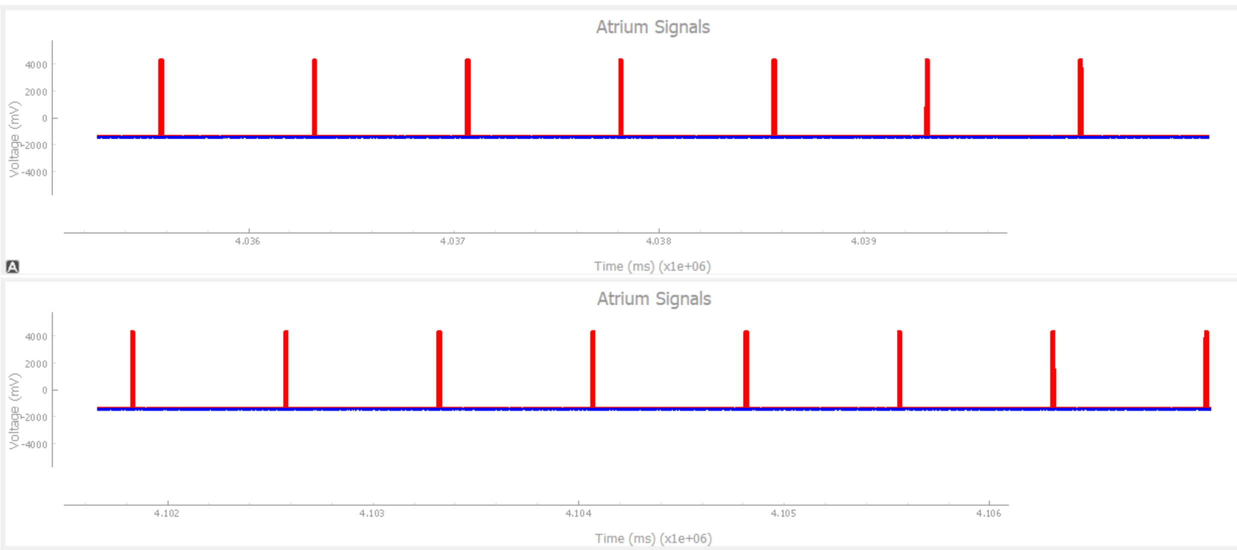
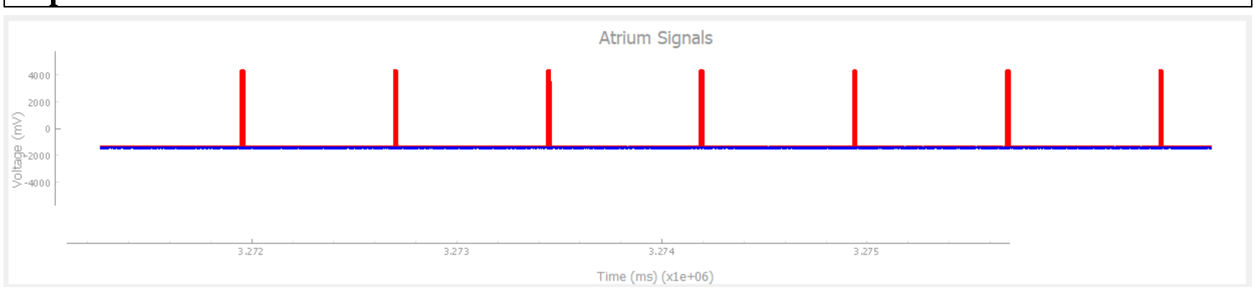


Figure 42: Atrium signal on HeartView™ for AAIR/VVIR test case #1

AAIR and VVIR: Test Case #2

Input



Pacemaker Settings		Heartview Settings	
Mode	AAIR	Natural Atrium	10 ms
Pulse Width (ms)	400	Natural Ventricle	Off
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	60
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
When the accelerometer is shaken, the pacemaker will elevate its pacing rate.		The result obtained shows that the atrium was not paced at all.	
Test result - *Failed*			
Change required – *The atrium should be paced.*			

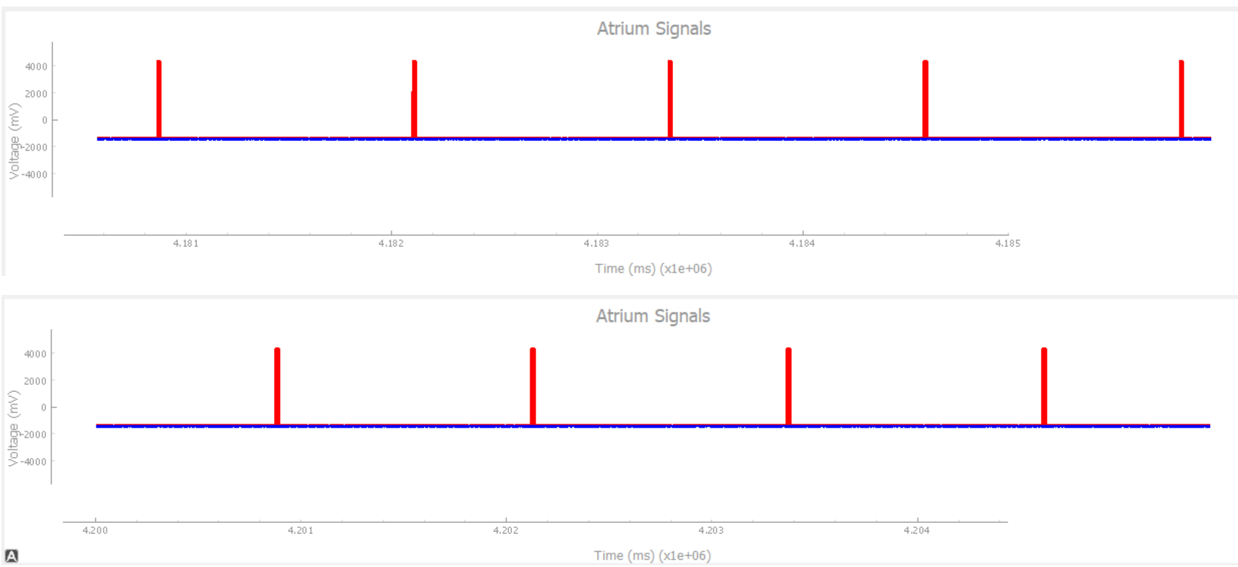


Figure 43: Atrium signal on HeartView™ for AAIR/VVIR test case #2

AAIR and VVIR: Test Case #3

Input			
Pacemaker Settings		Heartview Settings	
Mode	VVIR	Natural Atrium	Off
Pulse Width (ms)	400	Natural Ventricle	10 ms
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	100
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	

When the accelerometer is shaken, the pacemaker will elevate its pacing rate.	The result obtained shows that the ventricle was not paced when it was shaken.
Test result - *Failed*	
Change required – *The* ventricle *should be paced when shaken.*	

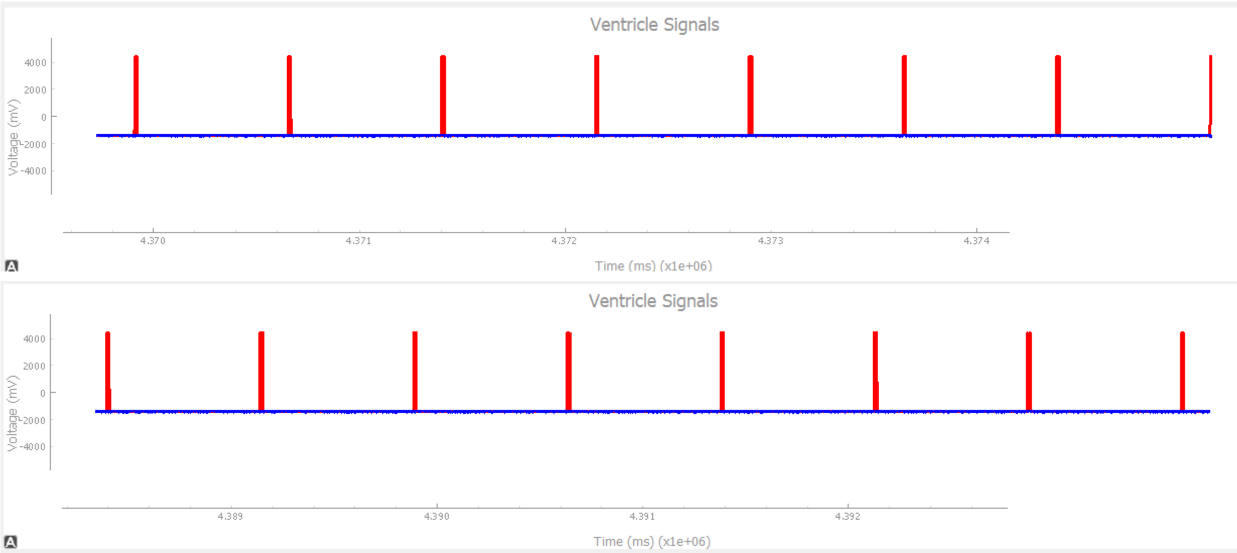


Figure 44: Venticle signal on HeartView™ for AAIR/VVIR test case #3

AAIR and VVIR: Test Case #4

Input			
Pacemaker Settings		Heartview Settings	
Mode	VVIR	Natural Atrium	Off
Pulse Width (ms)	400	Natural Ventricle	10 ms
Pulse Amplitude (V)	100	Natural Heart Rate (BPM)	60
Lower Rate (PPM)	100	Natural AV Delay (ms)	250
Output			
Expected		Actual	
When the accelerometer is shaken, the pacemaker will elevate its pacing rate.		The result obtained shows that the ventricle was not paced at all.	
Test result - *Failed*			
Change required – *The* ventricle *should be paced.*			

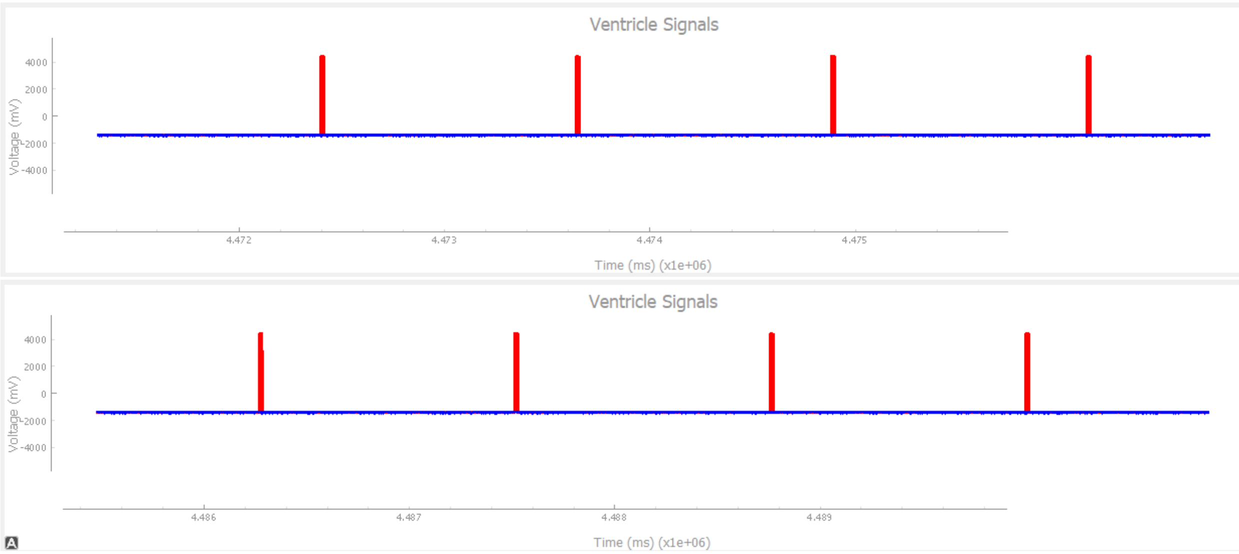
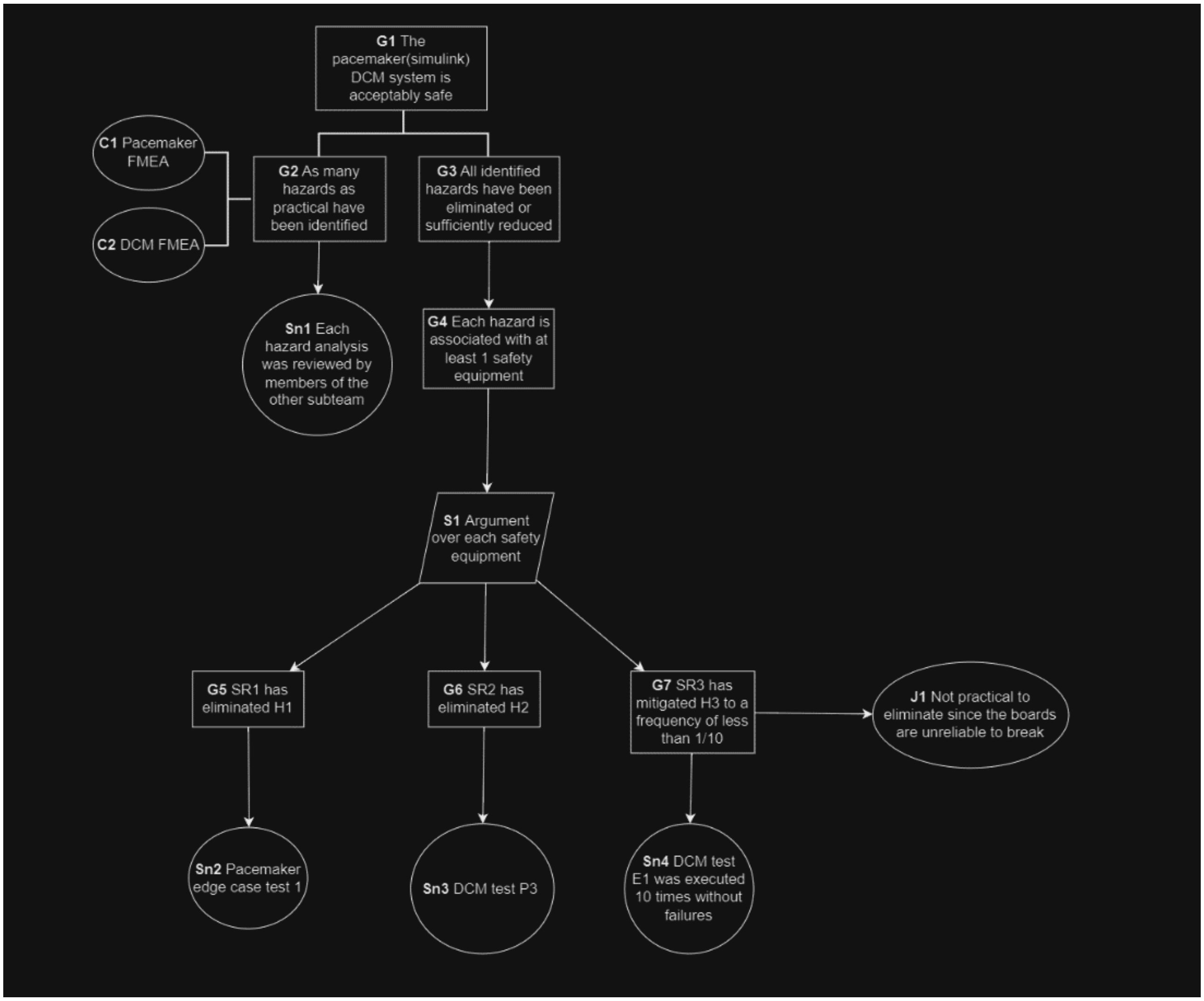


Figure 45: Venticle signal on HeartView™ for AAIR/VVIR test case #4

Assurance Case



Device Control-Monitor (DCM)

Testing (DCM)

ID	Purpose	Input	Expected Output
C1	Create User -> Normal case	Name = “Ninad” Password = “Rat” Confirm Password = “Rat”	User successfully created, does not display plain text password
C2	Create User -> Passwords not matching	Name = “Dirty” Password = “Dog” Confirm Password = “Rat”	Passwords mismatch, does not display plain text password
C3	Create User -> Null user case	Name = “” Password = “Siu” Confirm Password = “jok”	Invalid name/password, does not display plain text password
C4	Create User -> User duplicated	Name = “Ninad” Password = “Rat” Confirm Password = “Rat”	User already exists, does not display plain text password
C5	Create user -> Edge case -> Max number of users reached	10 users in database Name = “ujh” Password = “ujh” Confirm Password = “ujh”	System had reached to max number of users. Please contact support team for help, does not display plain text password
L1	Login -> Normal case	Name = “Ninad” Password = “Ninad”	Move to parameters screen, does not display plain text password
L2	Login -> No match	Name = “Ninad” Password = “jk”	Password incorrect or this user does not exist, does not display plain text password
P1	Parameters -> Normal case -> Test int and float parameters	Mode = AIIR Lower Rate Limit = 61 Atrial Amplitude = 4.0 All other parameters are their defaults	Parameters changed on UI to the ones inputted
P2	Parameters -> Edge case -> Test acceptance of int and float parameters at lower limit	Mode = DDDR for versions that have DDDR implemented, else AIIR All parameters set to the lowest accepted value	Parameters changed on UI to the ones inputted
P3	Parameters -> Edge case -> Test rejection of int and float parameters just past limit	Mode = AIIR Upper Rate Limit = H All other	Double check the value entered are in range for the parameter: Upper Rate Limit

		parameters are their defaults	
P4	Parameters -> Test null case	Mode = none	There is no submit parameters button or Please select a mode, mode can not be none
P5	Parameters -> Edge case -> Test acceptance of int and float parameters at upper limit	Mode = DDDR for versions that have DDDR implemented, else AIIR All parameters set to the highest accepted value	Parameters changed on UI to the ones inputted
E1	Egram	Press View Egram button	Displays graph of Egram data