Deliverable 1: Documentation Outline

Group #6

Fall 2025

Table of Contents

1 Group Members

Name	MacID	Student Number
Daniel Zhang	zhand40	400498813
Jay Seoh	Seohj1	400522306
Griffin Larke	larkeg	400497452
Dante	finorod	400449985
Muiz Hamzat	hamzatm	400500321
Bacem Karray	karrayb	400509045

2 Part 1

2.1 Introduction

The purpose of this pacemaker system is to monitor and regulate the heartrate of a patient. In particular, it is meant to support patients requiring bradycardia pacing. This is done by detecting the slow rhythms of the patient's heart and providing electrical pacing therapy to the heart.

This deliverable's main objective is to show ability to distinguish important information from provided documents to build the framework for a functional pacemaker and Device Controller-Monitor. This will be done by creating a design that satisfies any requirements provided by documentation.

The scope of this deliverable can be divided into two main components, pacemaker design along with DCM design.

For the pacemaker design, Simulink will be used to implement stateflow logic towards four pacing modes in permanent state: AOO, VOO, AAI, VVI. The design must follow the programmable parameters and requirements specified in the requirements document. Additionally, there must a "hardware hiding" subsection that correctly maps the pins from the pacemaker to the Simulink model. This abstracts the hardware from the design, leaving the stateflow design consistent even if the pinmap gets updated in the future.

Along with the pacemaker design, a DCM must be developed. For this deliverable, the DCM must include an interface that has a welcome screen with the ability to register up to 10 users, essential aspects of the user interface, interfaces to present all the pacing modes included in the current pacemaker design and making provisions to display and store key programmable parameters.

2.2 Requirements

The overall system for deliverable 1 must include:

- Monitoring and regulation of a patient's heart rate

- Sense the electrical activity of the heart
- Deliver electrical pacing therapy to the patient's heart
- Output pacing pulses with programmable pulse amplitudes and widths for atrium and ventricle
- Create a DCM that allows up to 10 users and presents all pacing modes

The pacemaker design for deliverable 1 must include:

- AOO (Atrial Asynchronous)
    - Asynchronous pacing mode
    - Paces delivered to the atrium at programmed Lower Rate Limit
    - Paces delivered without regard to senses
- VOO (Ventricular Asynchronous)
    - Asynchronous pacing mode
    - Paces will be delivered to the ventricle at programmed LRL
    - Paces delivered without regard to senses
- AAI (Atrial Inhibited)
    - Inhibited Pacing Mode
    - A sensed atrial event inhibits pending atrial pace and resets LRL timer
    - If no atrial event gets sensed, atrial pace gets delivered
    - Following an atrial event, the pacemaker shall enter an Atrial Refractory Period.
    - During ARP, any sensed atrial events shall not inhibit nor trigger pacing (do nothing)
- VVI (Ventricular Inhibited)
    - Inhibited Pacing Mode
    - A sensed ventricular event inhibits pending ventricular pace and resets LRL timer
    - If no ventricular event gets sensed, ventricular pace gets delivered
    - Following an atrial event, the pacemaker shall enter an Ventricular Refractory Period.
    - During VRP, any sensed atrial events shall not inhibit nor trigger pacing (do nothing)

2.3 Designs

The design for the pacemaker has the following characteristics:

System Architecture: The pins used in Deliverable 1 are as follow:

Sensing Interface:

- D13 (FRONTEND_CTRL): Control pin to enable sensing. must be set HIGH to enable sensing.
    - D0 (ATR_CMP_DETECT): Digital input pin for Atrial Sense. It outputs a HIGH signal when the atrial signal exceeds the set threshold.

    - D1 (VENT_CMP_DETECT): Digital input pin for Ventricular Sense. It outputs a HIGH signal when the ventricular signal exceeds the set threshold.
    - D6 (ATR_CMP_REF_PWM): PWM output pin used to set the atrial sensing threshold (sensitivity).
    - D3 (VENT_CMP_REF_PWM): PWM output pin used to set the ventricular sensing threshold (sensitivity)
- Pacing Interface (Charging State):
    - D5 (PACING_REF_PWM): PWM output pin that sets the desired pacing amplitude (voltage). The voltage on C22 is linearly proportional to the duty cycle of this PWM signal.
    - D2 (PACE_CHARGE_CTRL): Charges C22, when set to HIGH. Set to LOW during pacing state for safety. Prevents patient from being directly connected to PWM signal
    - D4 (Z_ATR_CTRL): Enables impedance circuit for atrium (connects to atrial ring). Set to low.
    - D7 (Z_VENT_CTRL): Enables impedance circuit for ventricle (connects to ventricular ring). Set to low
- Pacing Interface (Pacing State)
    - D10 (PACE_GND_CTRL): This pin is set HIGH to connect the electrode tip to ground, completing the pacing circuit.
    - For A_Pace Event: D8 (ATR_PACE_CTRL) is set HIGH to deliver the pace to the atrium. D9 (VENT_PACE_CTRL) is set to LOW.
    - For V_Pace Event: D9 (VENT_PACE_CTRL) is set HIGH to deliver the pace to the ventricle. D8 (ATR_PACE_CTRL) is set to LOW.
    - D4 (Z_ATR_CTRL): Enables impedance circuit for atrium (connects to atrial ring). Set to low.
    - D7 (Z_VENT_CTRL): Enables impedance circuit for ventricle (connects to ventricular ring). Set to low
- Recharge State:
    - D8 (ATR_PACE_CTRL) & D9 (VENT_PACE_CTRL): Both are set LOW.
    - D10 (PACE_GND_CTRL): This pin remains HIGH.
    - D11 (ATR_GND_CTRL): This pin is set HIGH (if the atrium was paced) to ground the atrial ring and allow C21 to discharge .
    - D12 (VENT_GND_CTRL): This pin is set HIGH (if the ventricle was paced) to ground the ventricular ring and allow C21 to discharge.

Programmable Parameters:

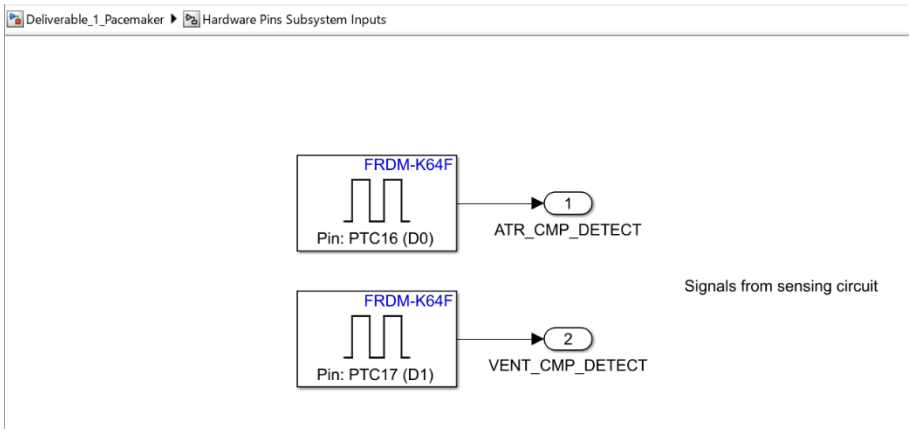
The following Parameters implemented during Pacemaker design are:

- Lower Rate Limit:

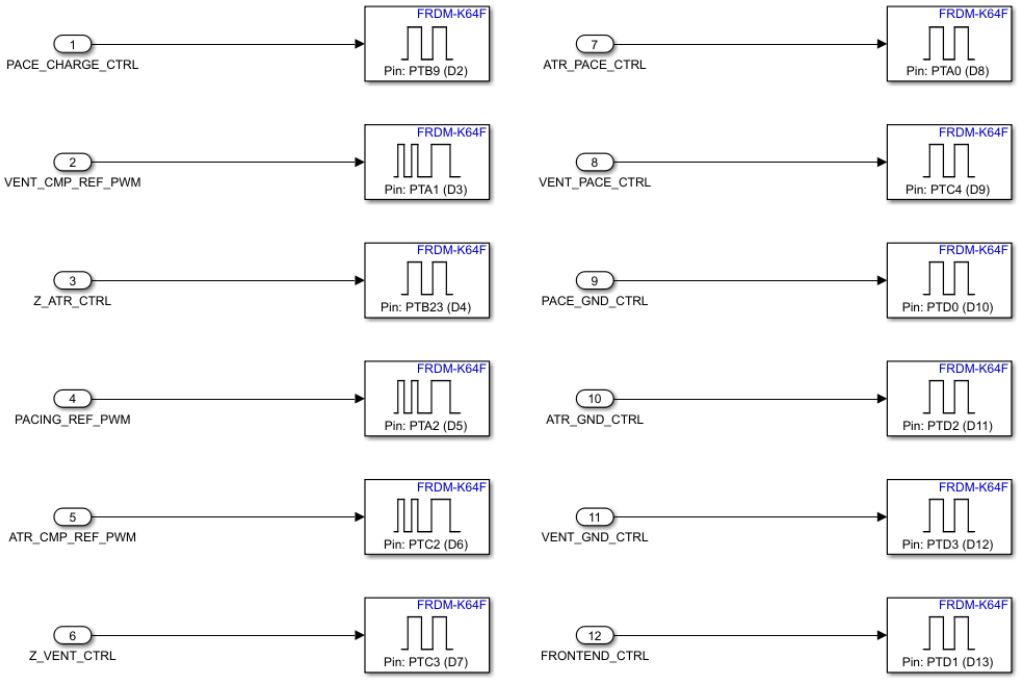
    - Number of generator pace pulses delivered per minute when there is no intrinsic activity or controlled pacing.
    - AOO, AAI, VOO, VVI
- Upper Rate Limit:
    - Maximum rate at which the paced ventricular rate will track atrial events, minimum time per pace.
    - AOO, AAI, VOO, VVI
- Atrial Amplitude:
    - Amplitude of the atrial signal
    - AOO, AAI
- Ventricular Amplitude:
    - Amplitude of the ventricular signal
    - VOO, VVI
- Atrial Pulse Width:
    - The duration of time to send an atrial pulse.
    - AOO, AAI
- Ventricular Pulse Width:
    - The duration of time to send a ventricular pulse.
    - VOO, VVI
- Atrial Sensitivity:
    - The minimum amount of voltage required to sense a natural atrium pace.
    - AOO, AAI
- Ventricular Sensitivity:
    - The minimum amount of voltage required to sense a natural ventricle pace.
    - VOO, VVI
- VRP (Ventricular Refractory Period):
    - Time interval following a ventricular event during which the sensors will do nothing.
    - VVI
- ARP (Atrial Refractory Period):
    - Time interval following a ventricular event during which the sensors will do nothing.
    - AAI
- PVARP:
    - Time interval following a ventricular event when an atrial cardiac event does not inhibit an atrial pace and trigger a ventricular pace.
    - AAI
- Hysteresis:
    - A longer period that encourages self pacing during exercise and other activities by waiting longer to pace after senses.

    - AAI, VVI
- Rate Smoothing:
    - Limits the pacing rate change due to precipitous changes to the intrinsic rate through two rate smoothing parameters, Rate Smoothing Up (pace rate increase doesn't exceed Rate Smoothing Up %) and Rate Smoothing Down (pace rate decrease doesn't exceed the Rate Smoothing Down %)
    - AAI, VVI

Hardware Inputs and Outputs:

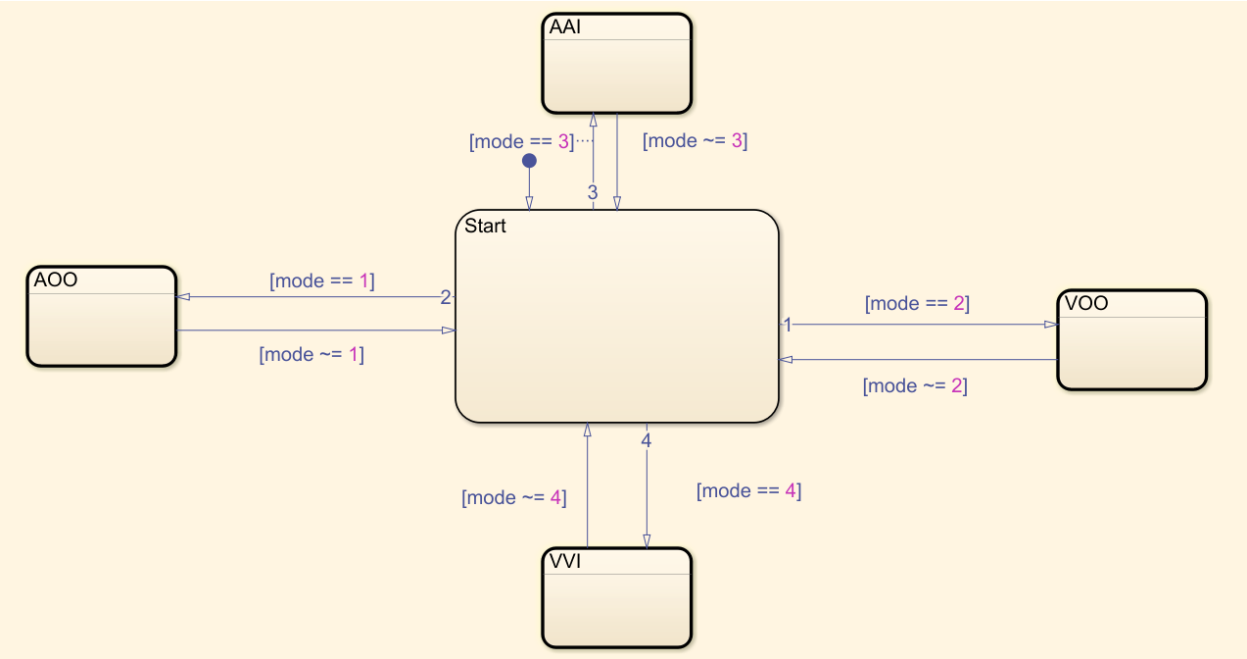


ATR_CMP_DETECT and VENT_CMP_DETECT are inputs that are digitally read from their respective hardware pins.



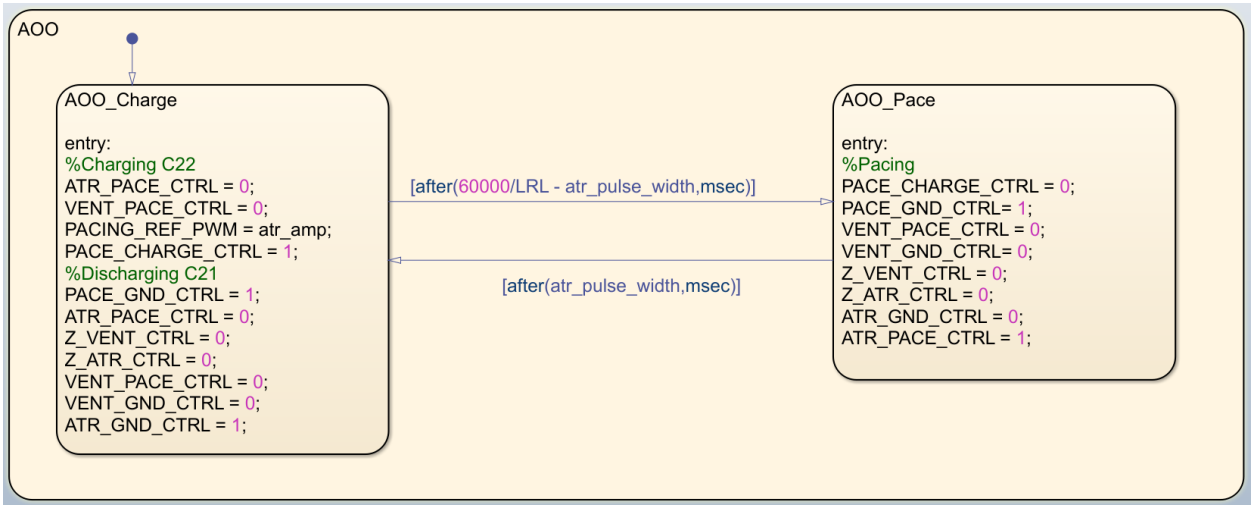
With the exceptions of VENT_CMP_REF_PWM, PACING_REF_PWM, and ATR_CMP_REF_PWM, all other outputs are digitally written to their respective hardware pins. The three outputs mentioned above are PWM outputs to their respective hardware pins.

State Machine Design (for each pacing mode):

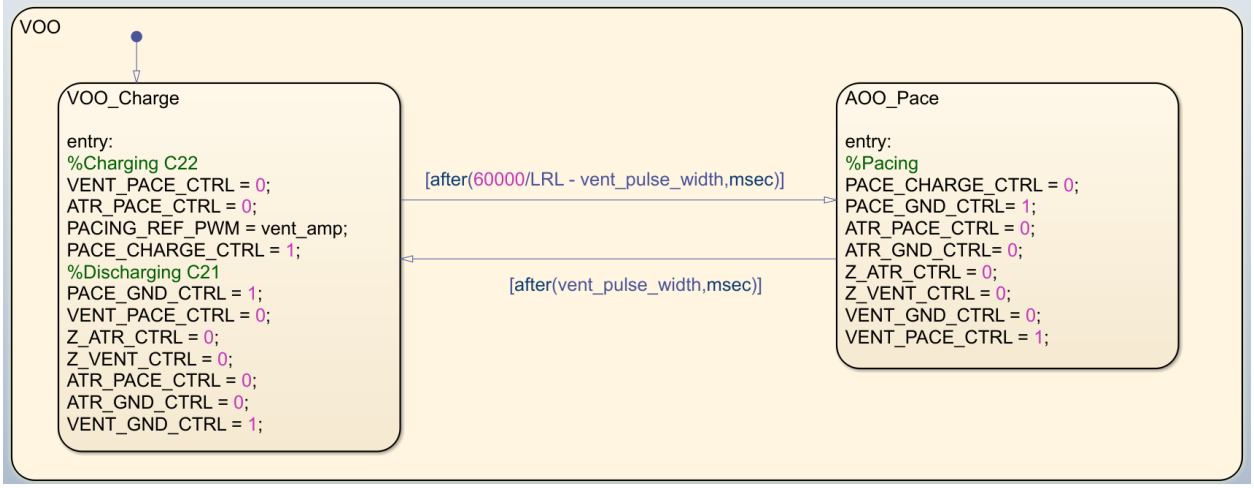


Depending on what the mode input is set to, the pacemaker logic will operate in either AOO, VOO, AAI, or VVI mode.

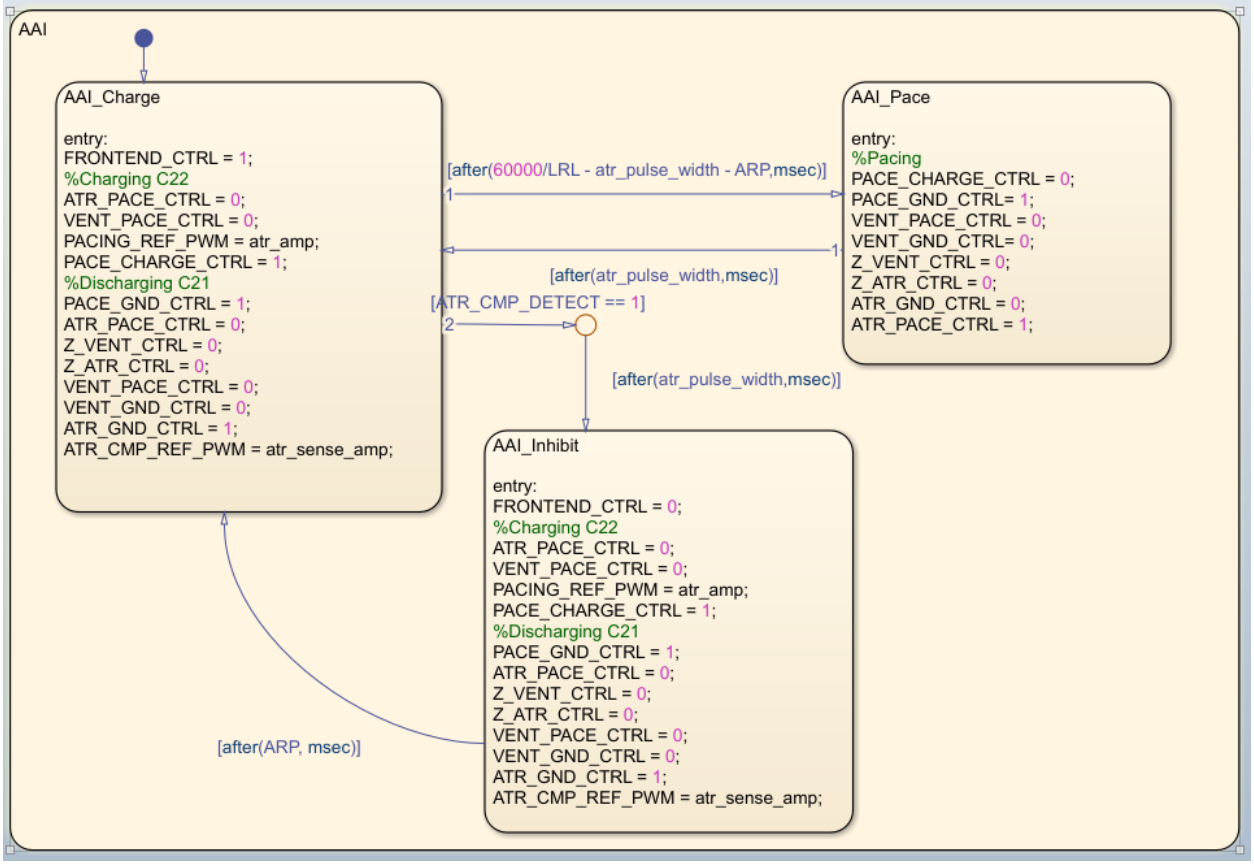
- AOO: The pacemaker will continuously switch between charging the primary capacitor (while discharging the blocking capacitor) and pacing the atrium



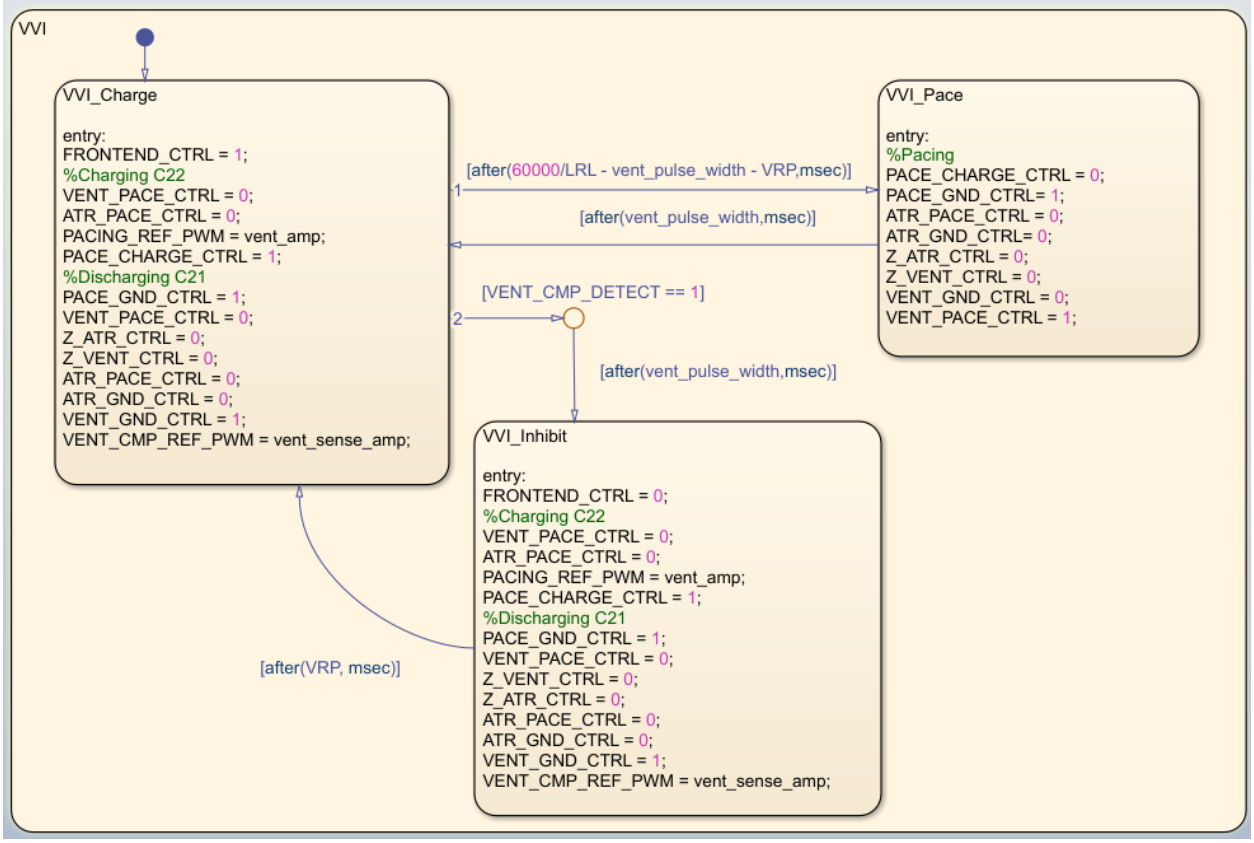
- VOO: The pacemaker will continuously switch between charging the primary capacitor (while discharging the blocking capacitor) and pacing the ventricle



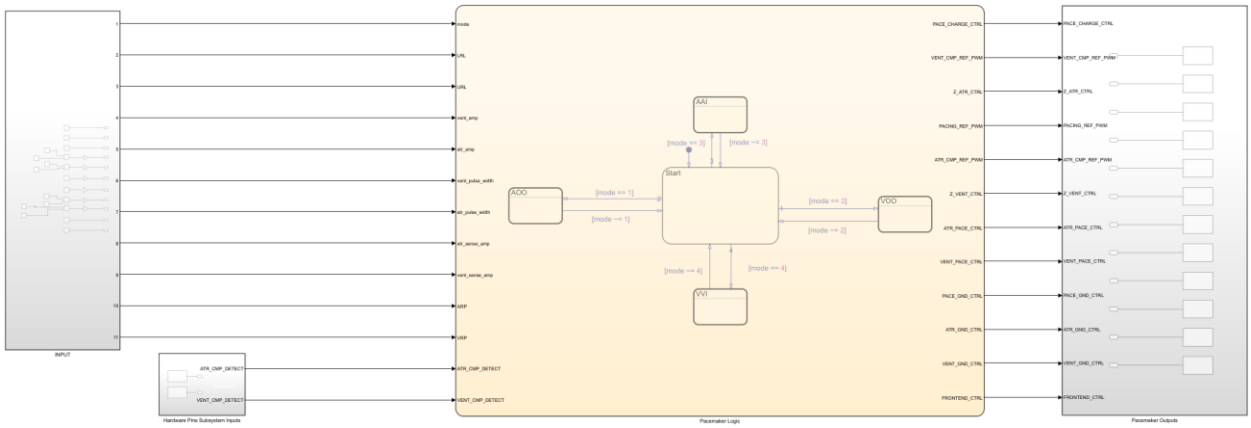
- AAI: Similar to AOO, but if the ATR_CMP_DETECT input detects a natural pulse from the atrium, the pacemaker will transition to an inhibit/buffer state instead of pacing the atrium, before transitioning back to the charging state



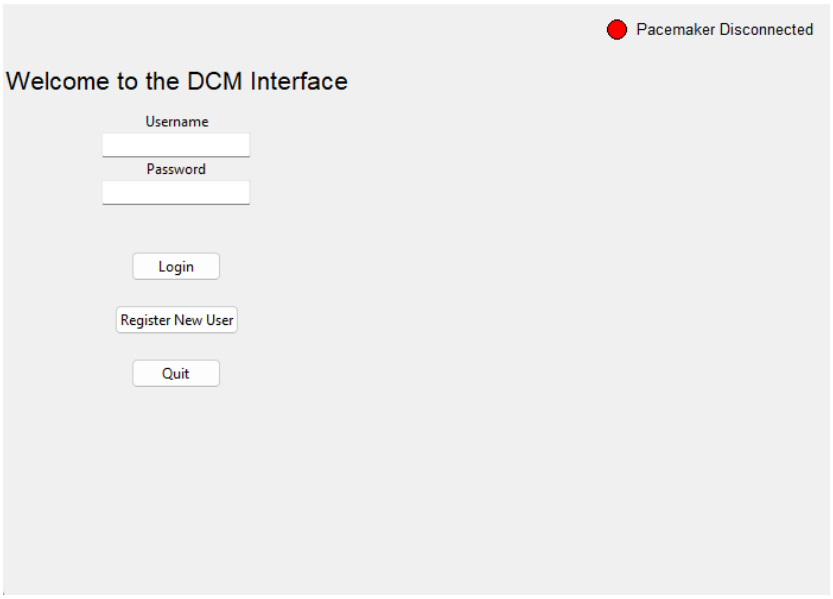
- VVI: Similar to VOO, but if the VENT_CMP_DETECT input detects a natural pulse from the ventricle, the pacemaker will transition to an inhibit/buffer state instead of pacing the ventricle, before transitioning back to the charging state



Simulink Diagram:

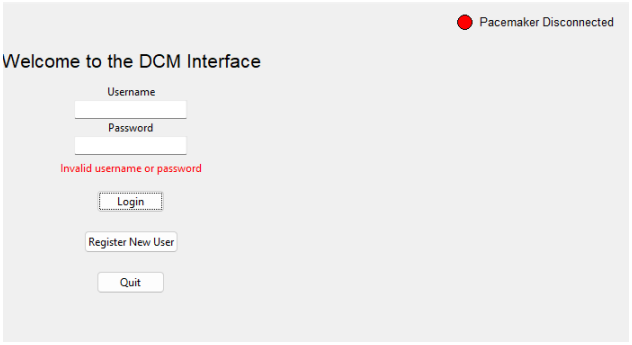


DCM structure:

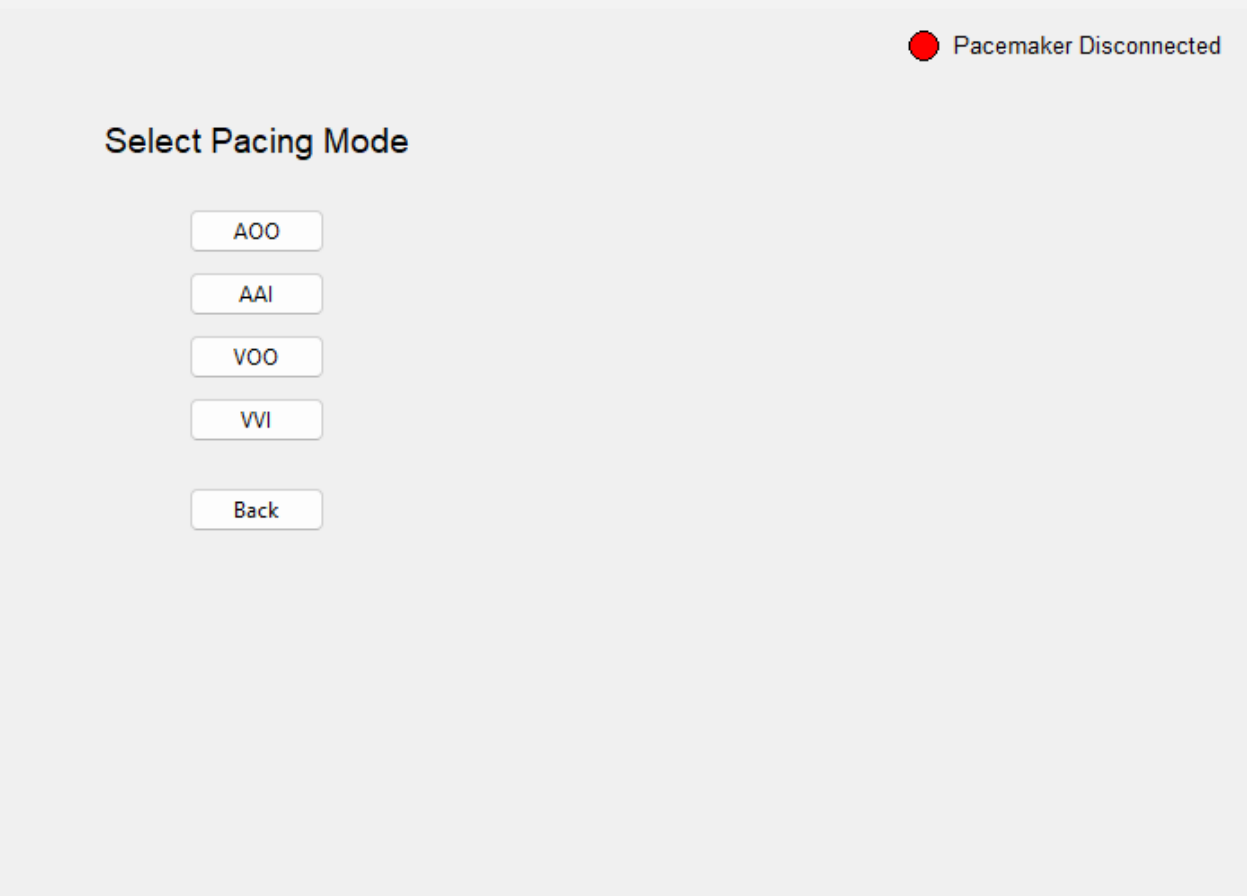


Welcome page:

This is the first page as seen by the user. It contains input fields for username and password, a login button, a new user registration button, and a quit button. If the login button is pressed and the login credentials are not valid then a failure message is shown.

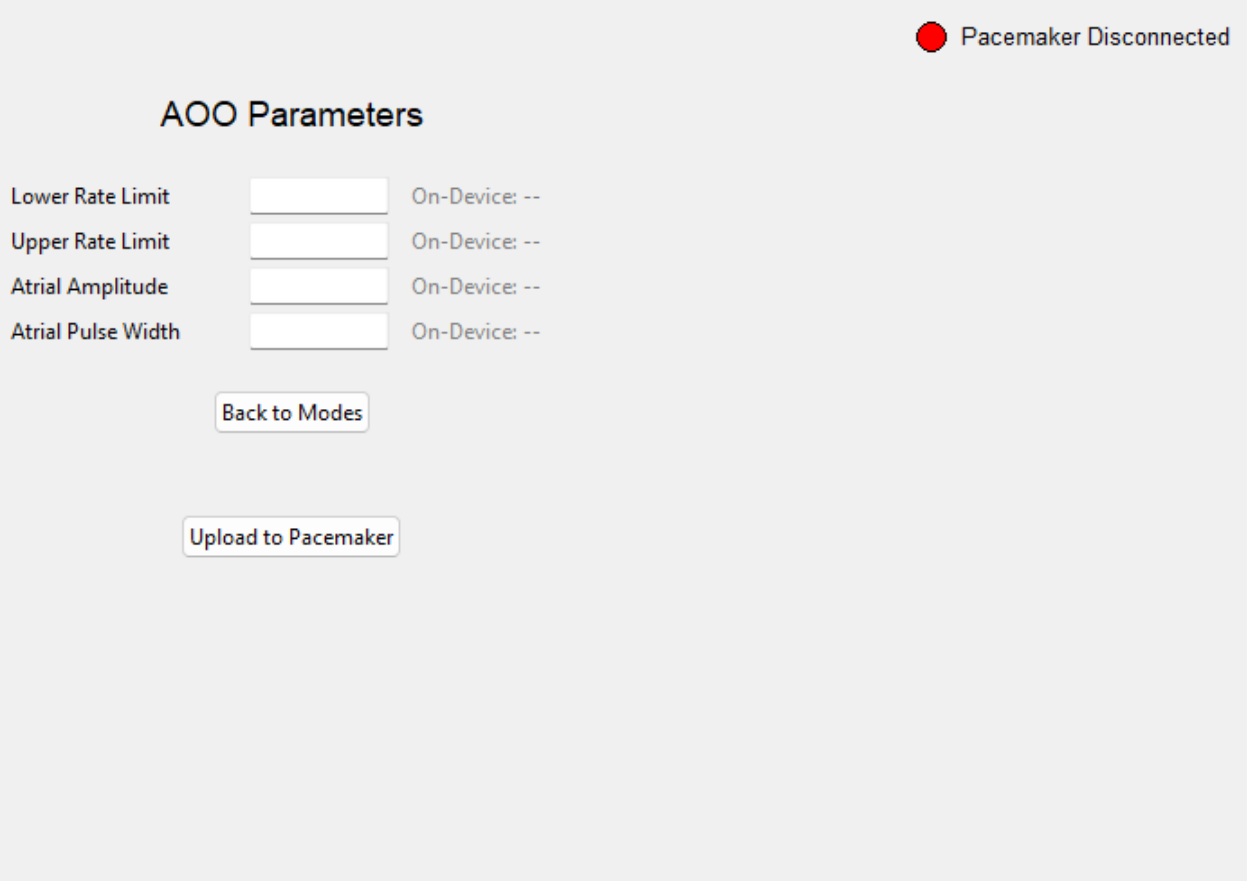


If the login credentials are valid then the user is brought to the mode select page. If the register new user button is pressed then the user is brought to the user registration page, and if the quit button is pressed then the program will exit.



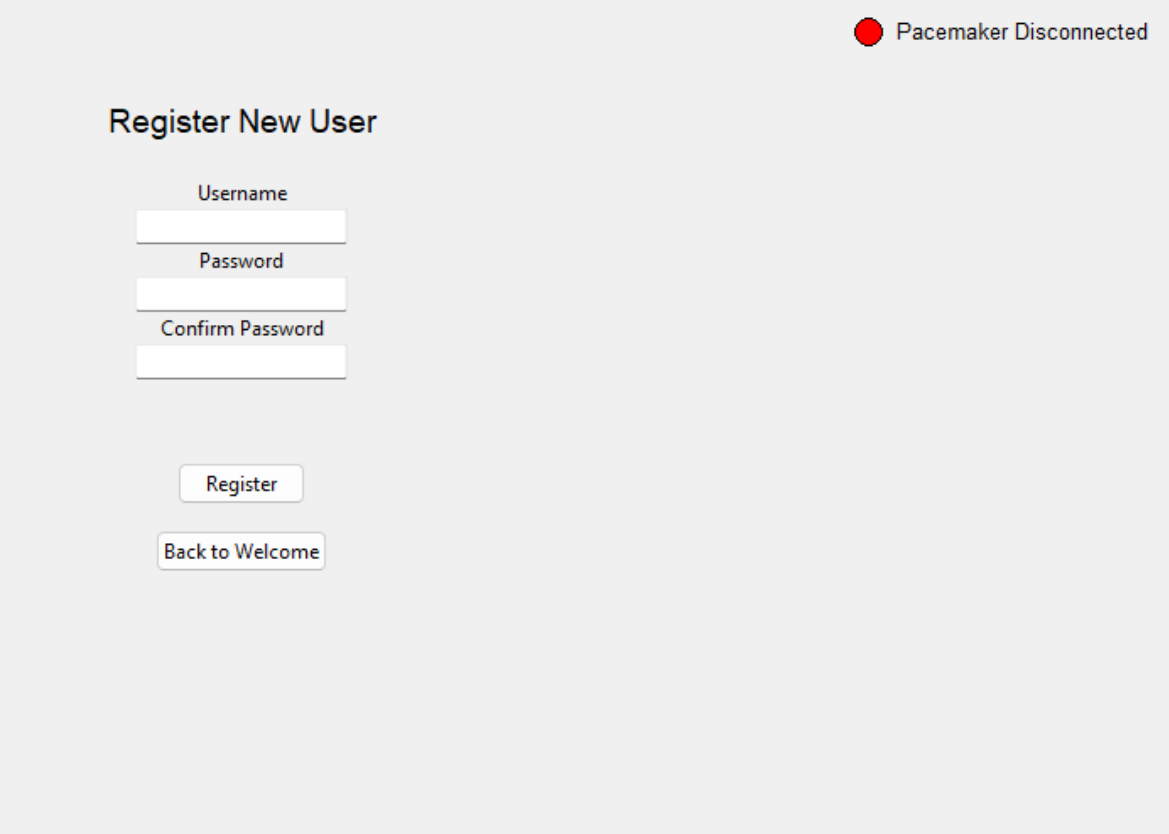
Mode select page:

This page displays the different pacing modes available to the user, with a button to go back to the welcome page, and buttons to go to the parameter page.



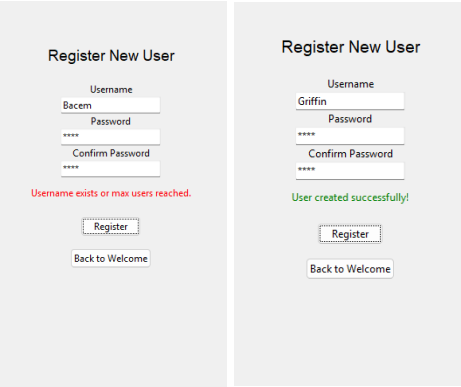
Parameter page:

This page is the same for every mode, except for the text displayed at the top. It allows the user to input upper and lower rate limits, as well as atrial amplitude and pulse width. It has a button to upload the inputted parameters to the pacemaker, and a button to return to the mode select screen. This page also text to distinguish between different pacemaker devices.



Register user page:

This page is shown if the user presses the register new user button on the welcome page. It has fields for username, password, and password confirmation. If the username is not taken, the passwords match and there are less than 10 users in the system, then a new user is created. If not, then an error message is displayed.



A success message is displayed if a user is created successfully.

3 Part 2

3.1 Requirements Potential Change

Based on the current requirements and design decisions, there are several potential changes to the requirements of the pacemaker system that can be undertaken:

1. Additional modes must be implemented to the pacemaker for deliverable 2 including AOOR,VOOR, AAIR, and VVIR.
2. With new modes, being implemented, more parameters and variables may be added into both the Simulink and the DCM. May need to adjust previous 4 states in case of new parameters
3. In deliverable 2, Simulink must be integrated to DCM in order to change Lower Rate Limit and Upper Rate limit along with different user databases.
4. Adjusting the rate of the pulses, should also include rate smoothing (I.E, an adapted rate)
5. In the DCM, there is currently no programmed way to communicate serially with a device. The only thing that exists is an indicator that no device is connected. This will certainly be expanded on in Deliverable 2. There will also need to be an established agreement on how the data will be sent, so it can be handled correctly when received.
6. The DCM will require a method to store and display parameters received from the pacemaker. This can be implemented easily if the UART connection is established next.
7. Currently, all inputted parameters in the DCM have no limits set to them because they can't be uploaded to the pacemaker yet. This will need to change when communication becomes possible.

3.2 Design Decision Potential Changes

Based on the current requirements and design decisions, there are several potential changes to the design decisions of the pacemaker system that can be undertaken:

1. Changing inputs from Simulink to be inputs from DCM rather than just constants
2. Will need to include UI for other the additional modes in the pacemaker
3. The DCM UI was created using Tkinter, a GUI toolkit. Despite taking an object oriented approach to the DCM, using Tkinter means that most often the UI components are created directly alongside the logic they trigger. That will make progress beyond Deliverable 1 more difficult, as the DCM will get a lot more sophisticated. Not just that, but it makes collaboration more difficult, as group members are more prone to merge conflicts and are heavily dependent on each other's progression. A potential solution to this would be using an alternative library, PyQt5. This would let us design a GUI instead of manually coding all elements. This makes it so that we don't have to spend nearly as much time and effort trying to program a button position and instead can focus on what

matters. Also, by refactoring the DCM code to integrate with PyQt5, the separation of work becomes much simpler, as the UI code would no longer be so tightly packed with the logic.

3.3 Module Description

Module 1 (Main Class):

Purpose:

- The main class serves as the core controller and entry point for the DCM Interface GUI application. It initializes the main application window, manages navigation between different pages/frames, and maintains global communication status indicators for the pacemaker device.

Key Functions and Methods

Function	Access	Description
__init__(self)	Public	Constructor that initializes main Tkinter window, sets up interface layout, initializes connection status indicator, creates/stores all application pages, displays default page.
show_frame(self, page_class)	Public	Switches the visible frame to the one specified by page_class. For navigation between different pages of the interface.
set_connection_status(self, connected: bool)	Public	Updates the connection status of the pacemaker. Displays a circle in the top corner of the interface, green if connected, red if disconnected.

Global and State Variables

Variable	Scope	Type	Description
status_frame	Instance	tk.Frame	Container for the connection status and label
status_canvas	Instance	tk.Canvas	Displays a visual status indicator for the pacemaker connection state.
status_label	Instance	ttk.Label	Text label that describes the current connection status.
frames	Instance	dict	Stores references to all application frames, keyed by reference type. Allows for page switching.

Interactions with Other Components

With Page Module:

- Creates and manages all GUI page instances.
- Provides central controller reference used by each page to switch views or access data.

Module 2 (Page Classes):

Purpose:

- This module defines the individual GUI pages used in the DCM interface. Each page inherits from tk.Frame and is managed by the main class. The pages handle specific user interface functionality like login, user registration, mode selection and parameter configuration.

Welcome page:

Key methods

Method	Access	Description
__init__(self, parent, controller)	Public	Initializes the welcome page layout, including username and password input, new user registration, and quit button. Also includes a message label for incorrect login attempts.
login_user(self)	Public	Checks the inputted user credentials against the user_db. If successful, logs the user in and goes to ModeSelectPage.
go_register(self)	Public	Switches display page to the RegisterUserPage.

Global and State Variables

Variable	Scope	Type	Description
controller	Instance	Reference	Reference the main application controller (Main).
username_entry	Instance	ttk.Entry	Input for username.
password_entry	Instance	ttk.Entry	Input for password.
message_label	Instance	ttk.Label	Displays message for login attempts.

Register User Page:

Key methods

Method	Access	Description
__init__(self, parent, controller)	Public	Initializes the registration page layout, has fields for new username, password, and password confirmation. Includes a button for registration and returning to the welcome page.
register_user(self)	Public	Validates password confirmation matches password, and attempts to create a new user via user_db module.
go_back(self)	Public	Returns to welcome page.

Global and State Variables

Variable	Scope	Type	Description
controller	Instance	Reference	Reference the main application controller (Main).
username_entry	Instance	ttk.Entry	Input for username.
password_entry	Instance	ttk.Entry	Input for password.
confirm_entry	Instance	ttk.Entry	Input for password confirmation.
message_label	Instance	ttk.Label	Displays registration feedback messages.

Mode Select Page:

Key methods

Method	Access	Description
__init__(self, parent, controller)	Public	Initializes the page layout, with buttons for each pacing mode and a button to return to the welcome page.
select_mode(self, mode)	Public	Sets the current pacing mode in the main controller and transitions to the parameter page.

Global and State Variables

Variable	Scope	Type	Description
controller	Instance	Reference	Reference the main application controller (Main).

Parameter Page:

Key methods

Method	Access	Description
__init__(self, parent, controller)	Public	Initializes the parameter page layout, with input boxes, buttons for navigating and uploading, and a message label for upload feedback.
show_paramters(self)	Public	Generates list of pacing parameters based on selected mode (AOO, AII, etc.). Clears and repopulates the form when the mode changes.
upload_to_pacemaker(self)	Public	Checks for pacemaker connection and uploads values. Displays an error message if pacemaker is not connected. (This hasn't been implemented yet)

Global and State Variables

Variable	Scope	Type	Description
controller	Instance	Reference	Reference the main application controller (Main).
widgets	Instance	ttk.dict	Stores parameter entry widgets, keyed by parameter name.
title	Instance	ttk.Label	Displays current pacing mode title.
form_frame	Instance	ttk.Frame	Container for parameter input fields.
upload_button	Instance	ttk.button	Triggers upload process to pacemaker.

<u>Interactions with Other Components</u>

- With Main Class:
  - Each page receives reference to the shared controller.
- With user_db:
  - Welcome Page uses user_db to validate login credentials.
  - Register User page uses user_db to to create a new account.
- With Pacemaker Communications (Future integration):
  - Parameter page will eventually call this module to send pacing parameters when a connection exists.

Module 3 (user_db):

Purpose:

- This module manages user authentication and registration data. It handles interactions with the local user data file (users.json) including loading, saving, validating and registering users. The module provides a simple interface for control in the GUI.

Key Functions:

Function	Access	Description
load_users()	Public	Loads and returns all existing user records from users.json.
save_users(data)	Internal	Saves a list of users in the same format of users.json. Overwrites users.json with new inputted list.
register_user(username, password)	Public	Adds a new user to users.json if the maximum limit (10) is not reached, and the user does not already exist in users.json. Returns True if successful, False if not.
check_login(username, password)	Public	Checks if a user with provided username and password exists in users.json. Returns True if username and password are valid, False if not.

Global and State Variables

Variable	Scope	Type	Description
none	-	-	Module is stateless, data is stored in external file.

<u>Interactions with Other Components:</u>

- With Page Classes:
    - Welcome page calls this module to check login credentials.
    - Register user page calls this module to register new accounts.

Module 4 (egram_data):

Purpose:

- The egram module is designed to handle collection, storage and management of egram data. Although not currently active, this module will be used in future versions of the DCM interface.

Key Functions:

Function	Access	Description
__init__(self)	Public	Initializes an EgramData instance with empty lists for time data and signal values.
clear(self)	Public	Clears all stored data, resetting time and signal lists.

Global and State Variables

Variable	Scope	Type	Description
Time	Instance	list[float]	Stores time values corresponding to each Egram data point.
Values	Instance	list[float]	Stores voltage or amplitude values of Egram signal at each time step.
egram	Global	EgramData	A global instance of EgramData class.

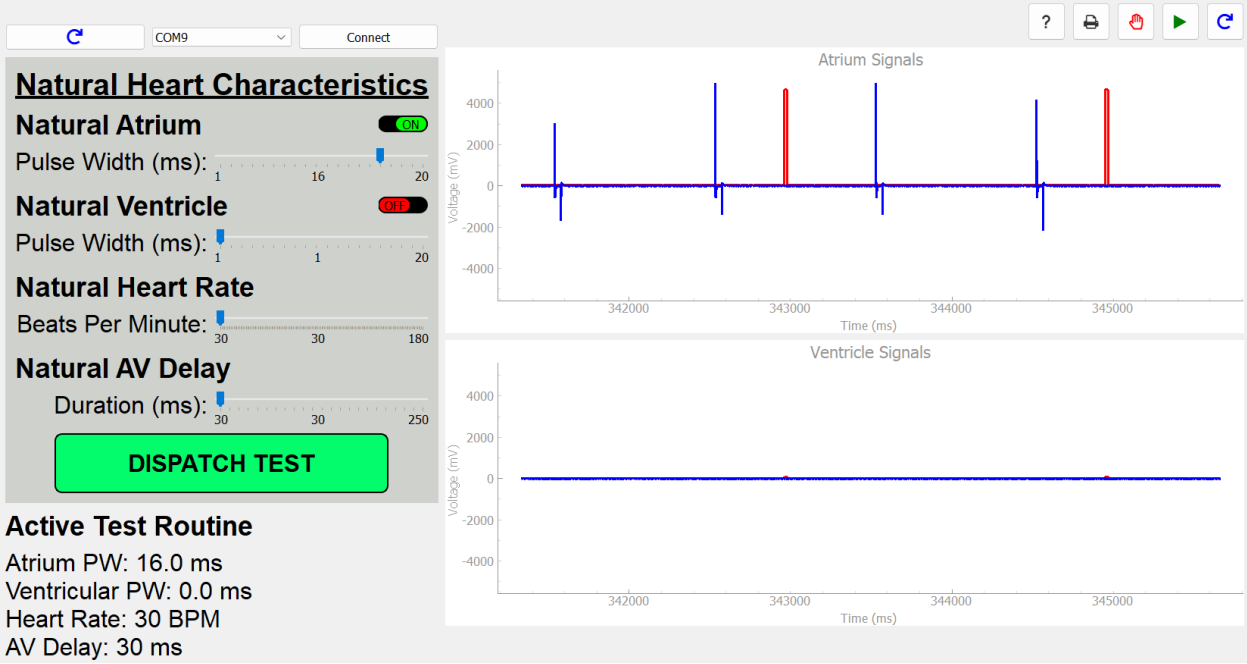
Interactions with Other Components:

- This module is not yet implemented

3.4 Testing

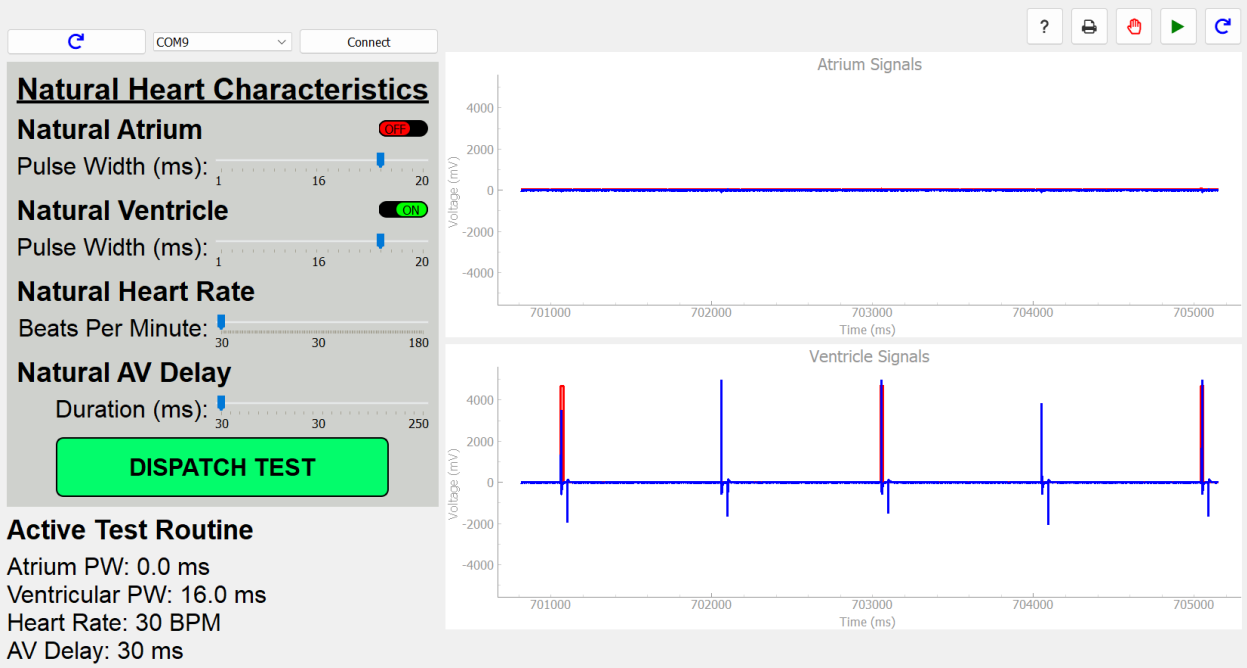
Simulink Testing: Simulink testing was done by watching the output from the pacemaker in Heartview and seeing how it reacted with natural heart pulses.

Mode 1: AOO



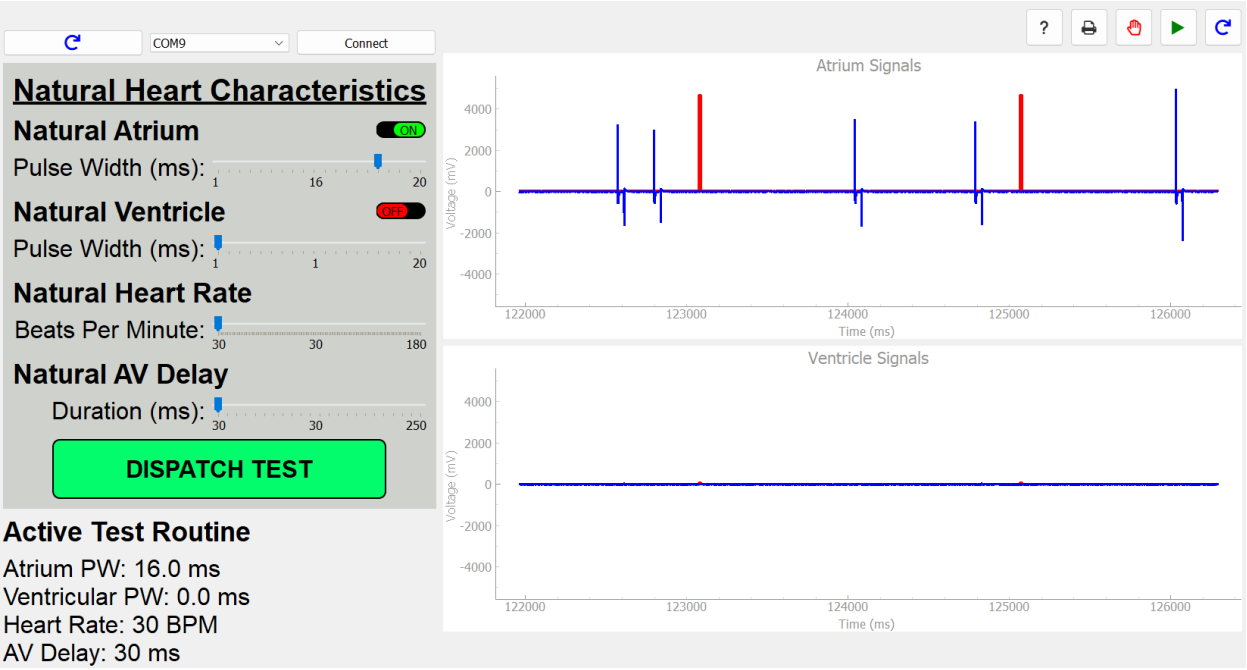
The pacemaker pulses to the atrium at the same rate, regardless of natural pulses.

Mode 2: VOO



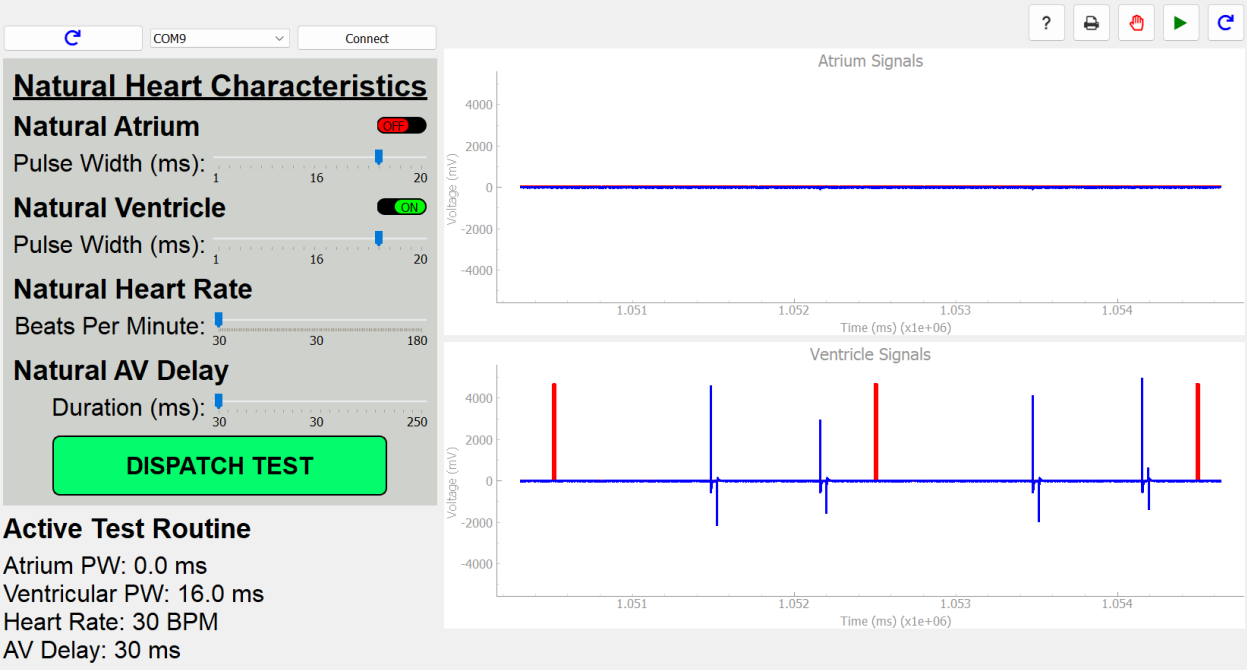
The pacemaker pulses to the ventricle at the same rate, regardless of natural pulses.

Mode 3: AAI



The pacemaker pulses to the atrium at the same rate, but will inhibit a pulse if there is a natural pulse.

Mode 4: VVI

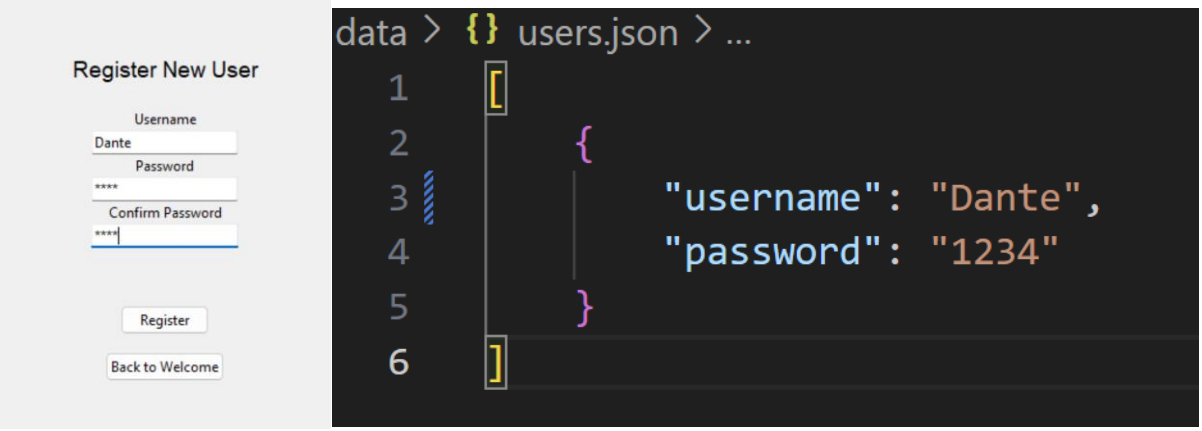


The pacemaker pulses to the ventricle at the same rate but will inhibit a pulse if there is a natural pulse.

DCM Testing

Test Case 1 (Register New user)

The purpose of this test is to see if we can create a new user profile. Given that there are no users already in out users.json data file we are going to open the DCM and follow the instructions to register a new user. It is expected that after we register a new user in the DCM, the new user information will appear in the user.json file.

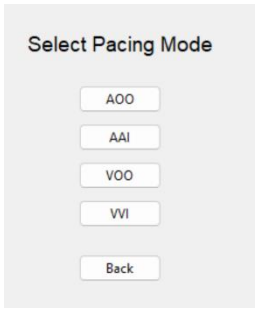


As you can see in figure above, after we registered a user, the information did indeed appear in the user.json file.

Result: Pass

Test case 2 (Logging in a user)

The purpose of this test is to confirm that a registered user can login. I am going to leave the "Dante", "1234" user in the user.json file and try and login with that username and password. The expectation is that the DCM should accept our login attempt and advance to the next menu.

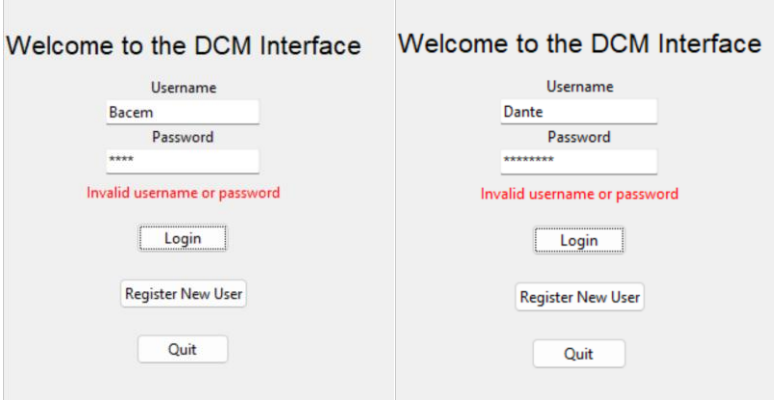


As you can see, when a know username and password is input it does proceed to the pacing mode menu.

Result: Pass

Test Case 3 (Invalid username or password)

The purpose of this test is to show that an invalid username or password is rejected, and an error message is shown. Leaving the user.json again with “Dante”, “1234”and attempting to login with “Dante”, “12345678” and “Bacem”, “1234” the expectation is that both attempts are rejected and the error message is shown.



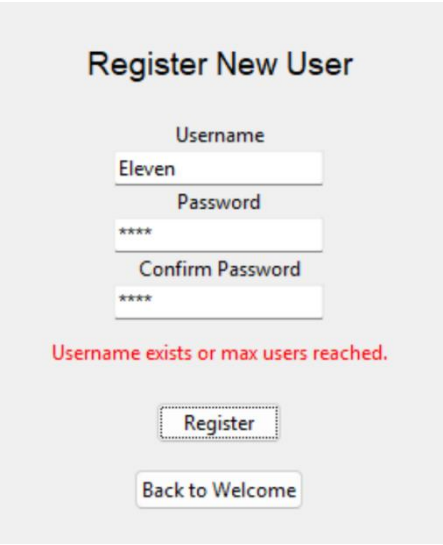
The attempts were rejected, and the error message was shown in both cases.

Result: Pass

Test Case 4 (Registering 11th user)

The DCM is supposed to hold 10 users at a time so the purpose of this test is to make sure an 11th user cannot be created. Here are the 10 users in the user.json file:

When an attempt is made to register a new user, it should be rejected and not added to the user.json file.



Here when the user “Eleven”, “1234” is attempted to be created it is rejected and not added to the user.json file.

Result: Pass

Test Case 5 (Registering a duplicate user)

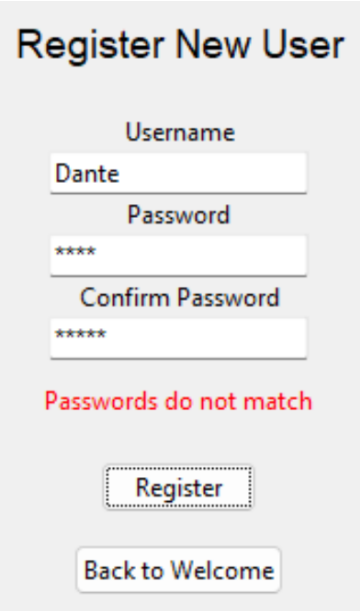
The purpose of this test is to prevent the creation of duplicate accounts. With "Dante", "1111" in the user.json file an attempt to make another account with "Dante" as the username will be made. This attempt should be rejected, the appropriate error message should be shown, and the second "Dante" username should not be added to the user.json file.

The attempt at creating a second "Dante" user was rejected.

Result: Pass

Test Case 6 (Passwords not matching)

When creating a new user the password needs to be entered twice to prevent a typo, so the purpose of this test is to verify that the passwords need to be matching, or the account will not be created. With an empty user.json file an attempt will be made to create the user "Dante", with the password being typed in first as "1234" and second as "12345". The request should be denied, the appropriate error message should come up and no user should be added to the user.json file.

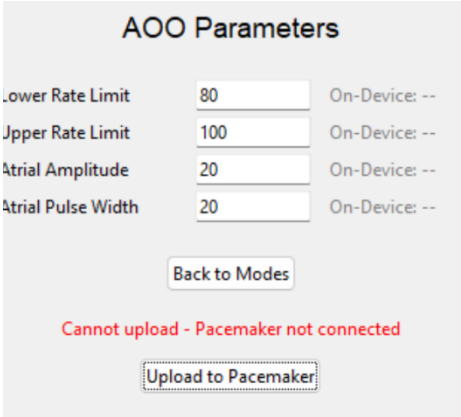


The attempt was denied.

Result: Pass

Test Case 7 (Uploading parameters to pacemaker)

This deliverable does not require connection to the pacemaker, so the purpose of this test is just to make sure the pacemaker parameters input boxes are functional. When an attempt to upload parameters to the pacemaker is made an error message should appear because the DCM is not connected to the pacemaker.



The error message did appear ensuring the input boxes and upload button is functional and ready to be fully implemented with connection to the pacemaker.

Result: Pass

3.5 GenAI Usage

Generative AI was used to help structure code in the DCM and with syntax and debugging. It was also used to help with formatting. No AI was used for Simulink stateflow design.

4 General Notes

Main class and page classes:

main_page.py > ...

```python
import tkinter as tk
from tkinter import ttk
import user_db

class Main(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("DCM Interface Demo")
        self.geometry("700x500")
        self.resizable(False, False)

        # Pacemaker status indicator. Exists globally.
        self.status_frame = tk.Frame(self)
        self.status_frame.pack(side="top", anchor="ne", padx=10, pady=10)
        self.status_canvas = tk.Canvas(self.status_frame, width=20, height=20, highlightthickness=0)
        self.status_canvas.pack(side="left")
        self.status_label = ttk.Label(self.status_frame, text="Pacemaker Disconnected", font=("Arial", 10))
        self.status_label.pack(side="left", padx=5)

        # By default
        self.pacemaker_connected = False
        self.set_connection_status(self.pacemaker_connected)

        # Container for all frames
        container = tk.Frame(self)
        container.pack(fill="both", expand=True)

        # Frames dictionary
        self.frames = {}

        # Append each page into the frames dict
        for F in (WelcomePage, ModeSelectPage, ParameterPage, RegisterUserPage):
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        # Show welcome page by default
        self.show_frame(WelcomePage)

    def show_frame(self, page_class):
        # bring the given frame to the front so that it is actually visible
        frame = self.frames[page_class]
        frame.tkraise()
        if page_class == ParameterPage:
            frame.show_parameters()

    def set_connection_status(self, connected: bool):
        self.pacemaker_connected = connected
        self.status_canvas.delete("all")
        if connected:
            self.status_canvas.create_oval(2, 2, 18, 18, fill="green")
            self.status_label.config(text="Pacemaker Connected")
        else:
            self.status_canvas.create_oval(2, 2, 18, 18, fill="red")
            self.status_label.config(text="Pacemaker Disconnected")
```

```python

# pages

class WelcomePage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        label = ttk.Label(self, text="Welcome to the DCM Interface", font=("Arial", 16))
        label.pack(pady=10)

        ttk.Label(self, text="Username").pack()
        self.username_entry = ttk.Entry(self)
        self.username_entry.pack()

        ttk.Label(self, text="Password").pack()
        self.password_entry = ttk.Entry(self, show="*")
        self.password_entry.pack()

        self.message_label = ttk.Label(self, text="", foreground="red")
        self.message_label.pack(pady=5)

        login_btn = ttk.Button(self, text="Login", command=self.login_user)
        login_btn.pack(pady=10)

        register_btn = ttk.Button(self, text="Register New User", command=self.go_register)
        register_btn.pack(pady=10)

        quit_btn = ttk.Button(self, text="Quit", command=controller.destroy)
        quit_btn.pack(pady=10)

    def login_user(self):
            username = self.username_entry.get().strip()
            password = self.password_entry.get().strip()

            if user_db.check_login(username, password):
                self.controller.current_user = username
                self.controller.show_frame(ModeSelectPage)
            else:
                self.message_label.config(text="Invalid username or password")

    def go_register(self):
        self.controller.show_frame(RegisterUserPage)

```

```python
class RegisterUserPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        ttk.Label(self, text="Register New User", font=("Arial", 14)).pack(pady=20)

        ttk.Label(self, text="Username").pack()
        self.username_entry = ttk.Entry(self)
        self.username_entry.pack()

        ttk.Label(self, text="Password").pack()
        self.password_entry = ttk.Entry(self, show="*")
        self.password_entry.pack()

        ttk.Label(self, text="Confirm Password").pack()
        self.confirm_entry = ttk.Entry(self, show="*")
        self.confirm_entry.pack()

        self.message_label = ttk.Label(self, text="")
        self.message_label.pack(pady=10)

        ttk.Button(self, text="Register", command=self.register_user).pack(pady=10)
        ttk.Button(self, text="Back to Welcome", command=self.go_back).pack(pady=5)

    def register_user(self):
        username = self.username_entry.get().strip()
        password = self.password_entry.get().strip()
        confirm = self.confirm_entry.get().strip()

        if password != confirm:
            self.message_label.config(text="Passwords do not match", foreground="red")
            return

        success = user_db.register_user(username, password)

        if success:
            self.message_label.config(text="User created successfully!", foreground="green")
        else:
            self.message_label.config(text="Username exists or max users reached.", foreground="red")

    def go_back(self):
        self.controller.show_frame(WelcomePage)
```

```python

class ModeSelectPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        ttk.Label(self, text="Select Pacing Mode", font=("Arial", 14)).pack(pady=20)

        # Used lambda to control when the select_mode function is called.
        # Without lambda's, the function calls immediately on program execution, not on button press
        ttk.Button(self, text="AOO", command=lambda: self.select_mode("AOO")).pack(pady=5)
        ttk.Button(self, text="AAI", command=lambda: self.select_mode("AAI")).pack(pady=5)
        ttk.Button(self, text="VOO", command=lambda: self.select_mode("VOO")).pack(pady=5)
        ttk.Button(self, text="VVI", command=lambda: self.select_mode("VVI")).pack(pady=5)

        ttk.Button(self, text="Back", command=lambda: controller.show_frame(WelcomePage)).pack(pady=20)

    def select_mode(self, mode):
        self.controller.current_mode = mode
        self.controller.show_frame(ParameterPage)



class ParameterPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.widgets = {}

        self.title = ttk.Label(self, text="", font=("Arial", 14))
        self.title.pack(pady=10)

        self.form_frame = tk.Frame(self)
        self.form_frame.pack(pady=10)

        self.back_button = ttk.Button(self, text="Back to Modes", command=self.go_back)
        self.back_button.pack(pady=10)

        self.upload_msg = ttk.Label(self, text="", foreground="red")
        self.upload_msg.pack(pady=5)

        self.upload_button = ttk.Button(self, text="Upload to Pacemaker",
                                        command=self.upload_to_pacemaker)
        self.upload_button.pack(pady=5)
```

```
192
193     def show_parameters(self):
194         # Needed to clear the frame if another mode has already been selected before
195         for widget in self.form_frame.winfo_children():
196             widget.destroy()
197         self.widgets.clear()
198
199         mode = self.controller.current_mode
200         self.title.config(text=f"{mode} Parameters")
201
202         # Mode-specific param lists
203         if mode == "AOO":
204             params = ["Lower Rate Limit", "Upper Rate Limit", "Atrial Amplitude", "Atrial Pulse Width"]
205         elif mode == "AAI":
206             params = ["Lower Rate Limit", "Upper Rate Limit", "Atrial Amplitude", "Atrial Pulse Width", "ARP"]
207         elif mode == "VOO":
208             params = ["Lower Rate Limit", "Upper Rate Limit", "Ventricular Amplitude", "Ventricular Pulse Width"]
209         elif mode == "VVI":
210             params = ["Lower Rate Limit", "Upper Rate Limit", "Ventricular Amplitude", "Ventricular Pulse Width", "VRP"]
211         else:
212             params = []
213
214         # Display entries
215         for p in params:
216             row = tk.Frame(self.form_frame)
217             row.pack(pady=2, fill="x")
218
219             ttk.Label(row, text=p, width=20, anchor="w").pack(side="left", padx=5)
220
221             entry = ttk.Entry(row, width=12)
222             entry.pack(side="left", padx=5)
223             self.widgets[p] = entry
224
225             # Placeholder for current pacemaker value until we actually program this (D2)
226             current_label = ttk.Label(row, text="On-Device: --", width=15, anchor="w", foreground="gray")
227             current_label.pack(side="left", padx=5)
228
229     def upload_to_pacemaker(self):
230         if not self.controller.pacemaker_connected:
231             self.upload_msg.config(text="Cannot upload - Pacemaker not connected", foreground="red")
232             return
233
234     def go_back(self):
235         self.controller.show_frame(ModeSelectPage)
236
237
238 if __name__ == "__main__":
239     app = Main()
240     app.mainloop()
```

user_db:

user_db.py > register_user

```python
import json


def load_users():
    with open("data/users.json", "r") as f:
        return json.load(f)


def save_users(data):
    with open("data/users.json", "w") as f:
        json.dump(data, f, indent=4)


def register_user(username, password):
    data = load_users()
    # Can't register multiple users with same username
    if len(data) == 10:
        return False
    for user in data:
        if user["username"] == username:
            return False

    data.append({
        "username": username,
        "password": password
        })
    save_users(data)
    return True


def check_login(username, password):
    data = load_users()
    for user in data:
        if user["username"] == username and user["password"] == password:
            return True
    return False

register_user("Bacem", "1234")

```

egram data (for future implementation:

egram_data.py > ...

```python
class EgramData:
    def __init__(self):
        self.time = []
        self.values = []

    # Method for wiping all the current egram data stored
    def clear(self):
        self.time.clear()
        self.values.clear()



egram = EgramData()

egram.time.append(40)
egram.values.append(40)
print(egram.time)
print(egram.values)
egram.clear()
print(egram.time)
print(egram.values)

```