# Assignment 1 Report

## Pacemaker Development

## MECHTRON 3K04


Michael Giancola
400372137

Fraser MacFarlane
400379429

Luai Bashar
400388669

Karm Desai
400372652

Nickolas Koenig
400370473

Matthew Mark
400373381


October 15, 2023

## *Table of Contents*

# Academic Integrity Statement

The students are responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with their names and student numbers is a statement and understanding that this work is their own and adheres to the Academic Integrity Policy of McMaster University.

| | |
|---|---|
| Luai Bashar | 400388669 |
| Michael Giancola | 400372137 |
| Nicholas Koenig | 400370473 |
| Matthew Mark | 400373381 |
| Karm Desai | 400372652 |
| Fraser MacFarlane | 400379429 |

# Introduction

A pacemaker is a medical device engineered to synchronize, sense, and deliver precise electrical impulses to the heart as necessary. These electrical signals, transmitted through strategically placed electrodes, allow for the contractions of the heart chambers, ensuring an optimal blood-pumping rhythm. By doing so, this device plays a pivotal role in regulating the heart's electrical conduction system to maintain a safe and healthy body. In the pursuit of advancing pacemaker technology, our project involves the design and development of a foundational pacemaker framework. Leveraging Simulink for design and Heart view for troubleshooting, we meticulously created state flow diagrams to ensure a clear operation. Furthermore, we have integrated a Device Controller-Monitor (DCM) featuring an interactive Graphical User Interface (GUI). This DCM not only provides comprehensive information about the pacemaker device it oversees but also facilitates clear communication to issue control instructions.

# Pacemaker

## Planning and Design

When approaching the pacemaker design, a crucial starting point was gaining a comprehensive understanding of the heart's anatomy and its intricate electrical conduction system. This foundational knowledge was essential to determine where the electrical stimuli should be administered. We also learned the specifications of the different pacing modes so that they could be programmed accordingly. For this assignment, we were required to implement 4 different modes which included VOO, AOO, VVI and AAI. Each of these have their own distinct operation, which is shown in more depth below.

| **Mode:** | **Function:** |
|---|---|
| VOO | Ventricular pacing, no sensing |
| AOO | Atrial pacing, no sensing |
| VVI | Ventricular pacing, ventricular sensing, sensed intrinsic signal inhibits ventricular pacing |
| AAI | Atrial pacing, atrial sensing, sensed intrinsic signal inhibits atrial pacing |

*Table 1: Pacemaker Modes Functionality*

## Requirements and Specifications

| Requirement | Specification |
|---|---|
| Pacemaker must be able to emit electrical stimuli which allows medical professionals to pulse heart safely. | Pacing modes must be precise in design and follow safety guidelines (i.e limits, delays, frequencies, etc). |
| Pacing modes to implement:<br>• AOO<br>• VOO<br>• AAI | Modes must operate as described within Pacemaker documentation (explained briefly within Table 1). |

| | |
|---|---|
| • VVI | |
| Heart and Pacemaker hardware must be able to communicate with one another to provide appropriate output when needed. | Pacing modes must be designed using Simulink software within Matlab and built within the NXP FRDM-K64F hardware where design shall be tested. |

*Table 2: Pacemaker Requirements/Specifications*

## States

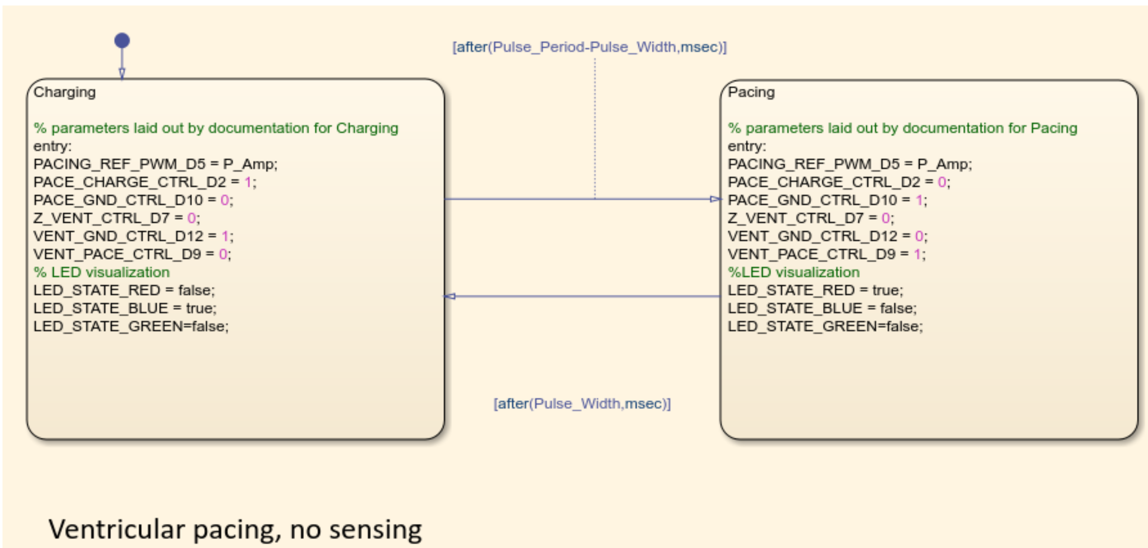



*Figure 1: VOO Stateflow*

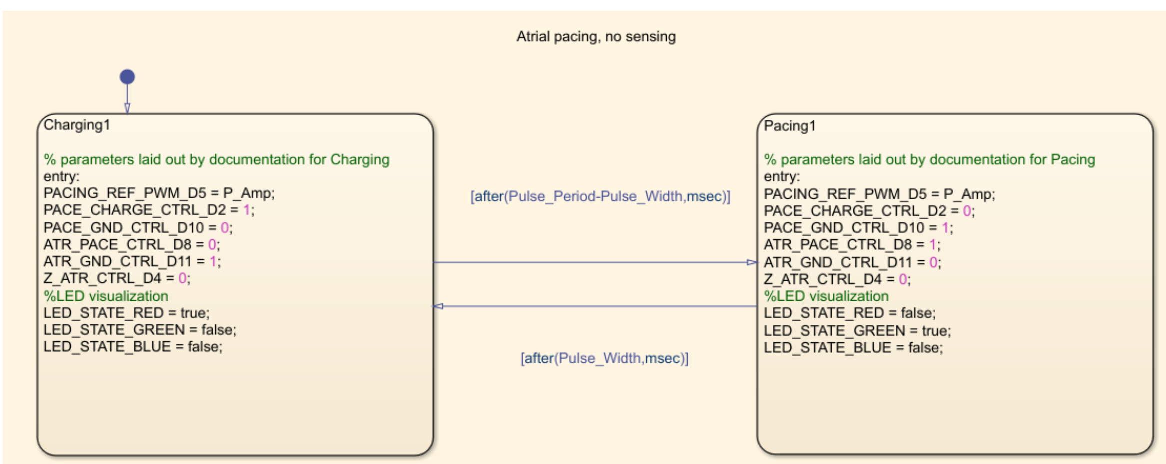



*Figure 2: AOO Stateflow*

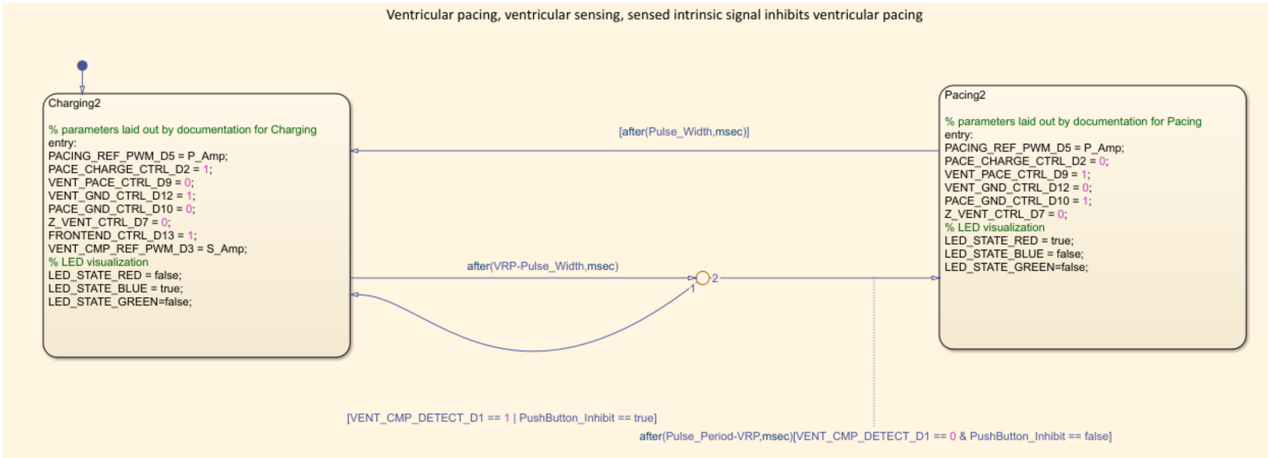



*Figure 3: VVI Stateflow*

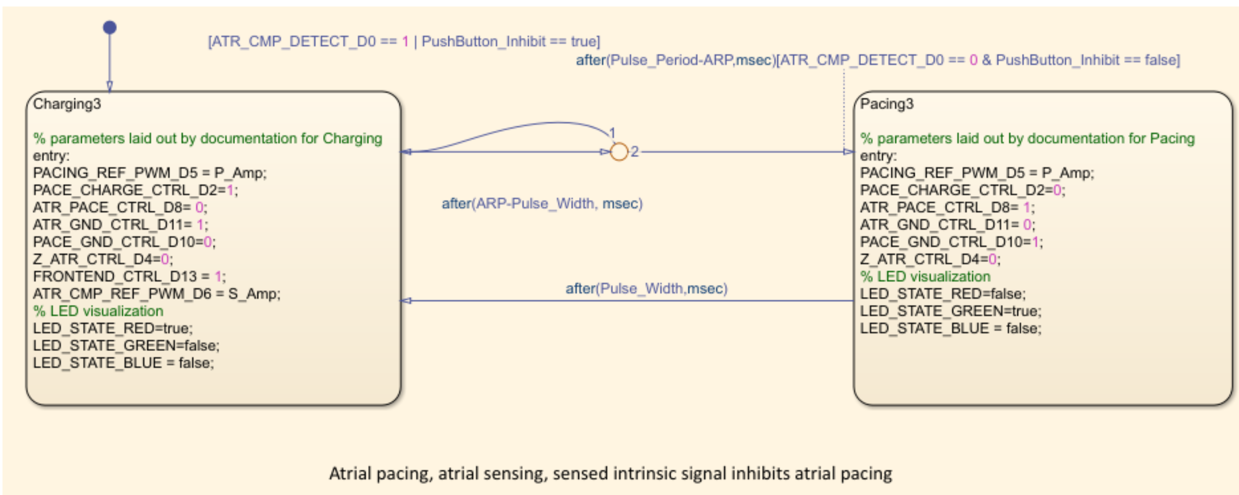



*Figure 4: AAI Stateflow*

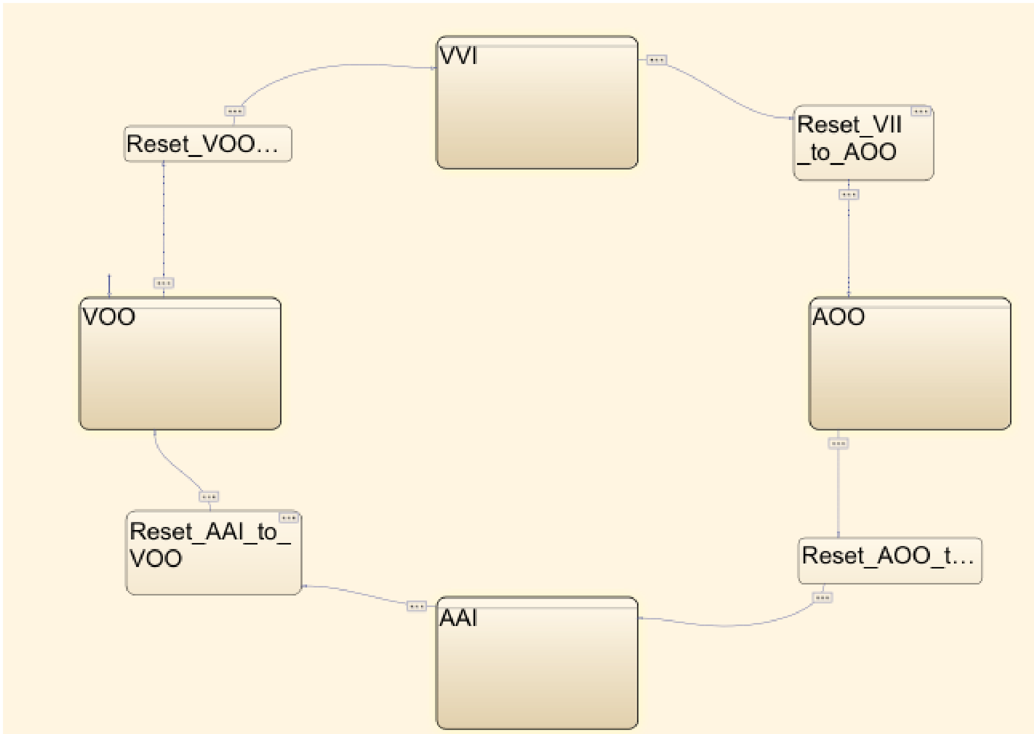



*Figure 5: Connected Modes*

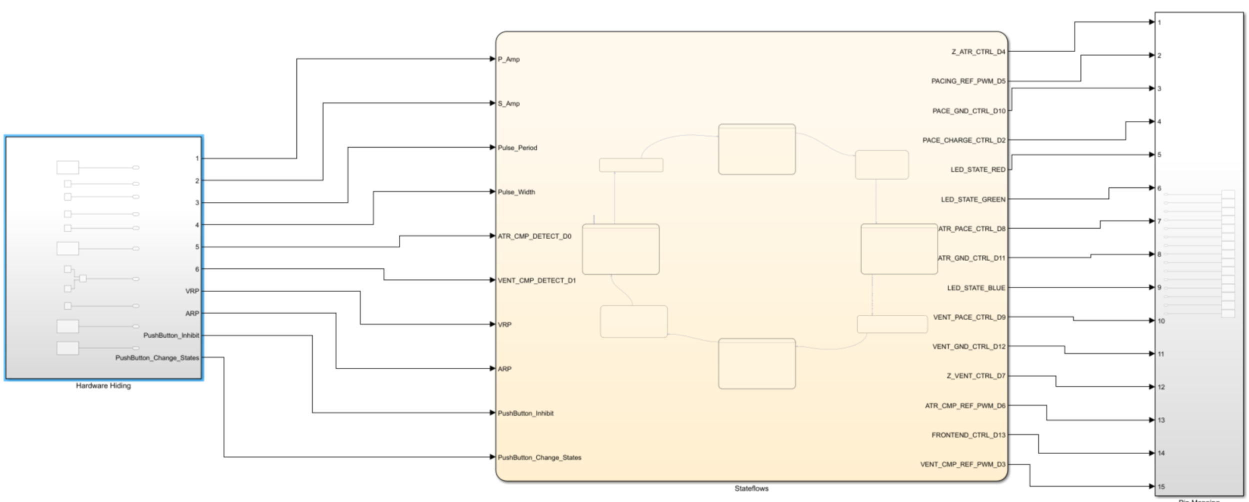



*Figure 6: Merged Stateflow with Input/Output*

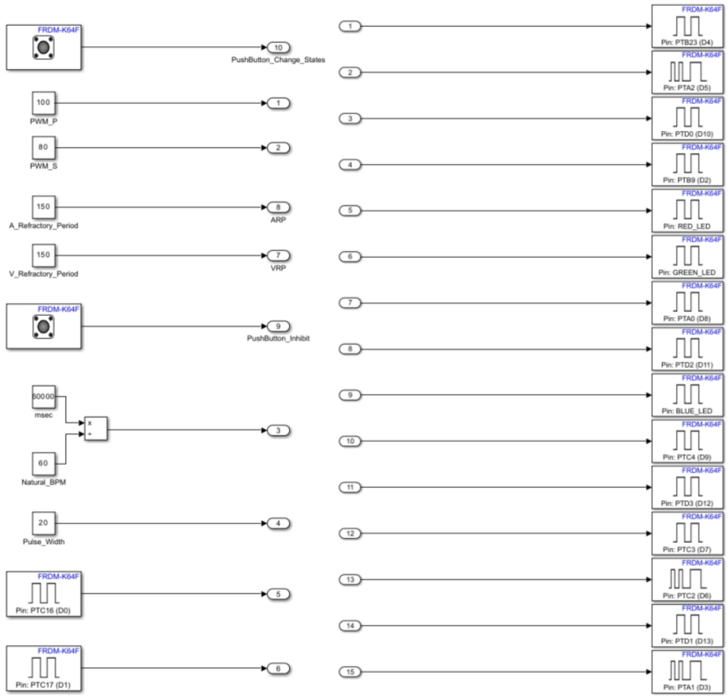



*Figure 7-8: Left - Input Pin Mapping, Right - Output Pin Mapping*

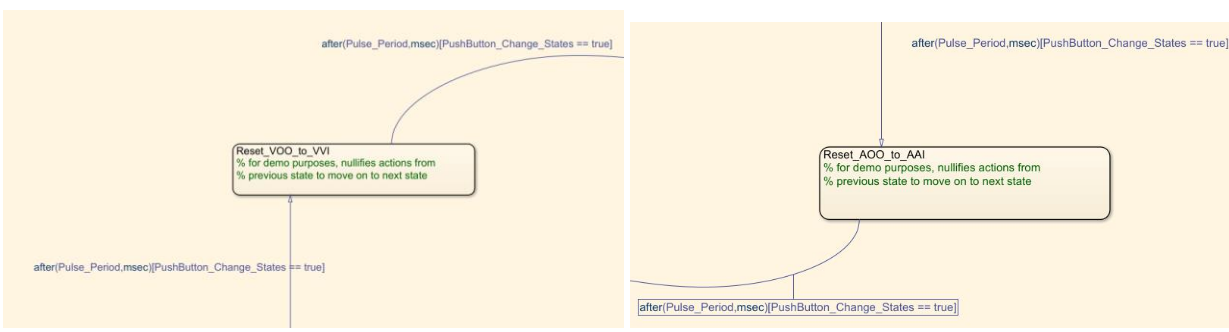



*Figure 9-10: Left – Reset State From VOO to VVI, Right - Reset State From AOO to AAI*

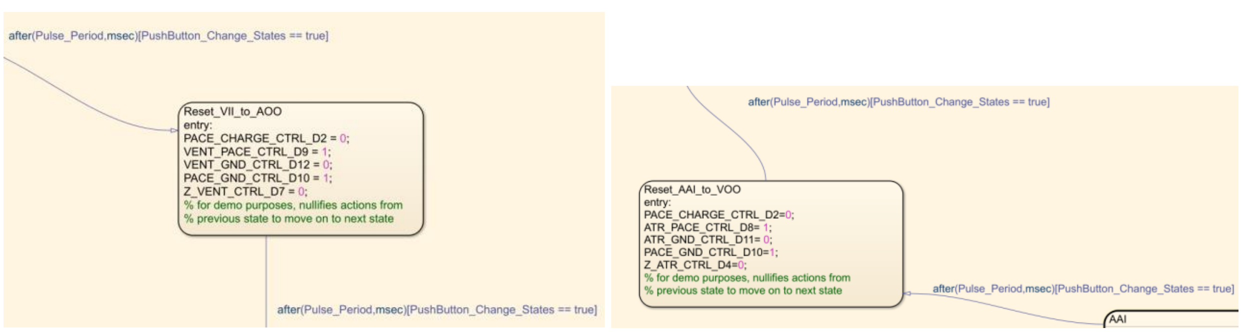



*Figure 11-12: Left – Reset State From VVI to AOO, Right - Reset State From AAI to VOO*

## States Description

| State: | Function: |
|---|---|
| Charging | Charges capacitor and continuously sensing while in this state |
| Pacing | Discharges capacitor and sets the pace |
| Reset | Changes states in firmware to avoid time it takes to re-build and deploy different states<br><br>*Note: Strictly for ease of demonstration of modes |

*Table 3: State functionality within modes*

## Monitored and Control Variables

**VOO:**

| Variables: | Initialization: | Function: | Description: |
|---|---|---|---|
| **Control:** | | | |
| PACING_REF_PWM_5 | P_Amp<br><br>= 100 [%] | Charges primary capacitor | Constant, represents the pacing PWM output that controls the amplitude of the pace |
| PACE_CHARGE_CTRL_D2 | 1 | Starts and stops PACING_REF_PWM_D5 (charging capacitor) | High in charging, low in pacing. This prevents the capacitor from charging while it's emitting the charge as a pulse to the ventricle. |
| PACE_GND_CTRL_D10 | 0 | Controls current flow to the tip of the ventricle from the ring | Low in charging, high in pacing. Prevents pulse from delivering too much charge to the ventricle. |
| Z_VENT_CTRL_D7 | 0 | Connects impedance circuit and ventricle ring electrode | Always set to low to prevent circuit from short circuiting. |
| VENT_GND_CTRL_D12 | 1 | Stops charge buildup during capacitor discharge | High in charging, low in pacing state. Prevents capacitor from overcharging |
| VENT_PACE_CTRL_D9 | 0 | Discharges primary capacitor through ventricle | Low in charging, high in pacing state. Allows capacitor to build up enough charge. |

| | | | |
|---|---|---|---|
| LED_STATE_RED<br>LED_STATE_BLUE<br>LED_STATE_GREEN | 0<br>1<br>0 | Indicates change in charging and pacing state | *Note: Implemented for demonstration purposes only |
| Pulse_Period | 60000 [msec/min] / 60 [BPM]<br><br>= 1000 [msec/beats] | Converts BPM to total time from the end of one pulse to the end of another pulse in msec | Responsible for transition timing between charging and pacing state. |
| Pulse_Width | 20 [msec] | Length of the pulse, discharging duration of capacitor in msec | Measure of time between single pulse of energy. |

*Table 4: VOO Variable Descriptions*

**AOO:**

| **Variables:** | **Initialization:** | **Function:** | **Description:** |
|---|---|---|---|
| **Control:** | | | |
| PACING_REF_PWM_D5 | P_Amp<br><br>= 100 [%] | Charges primary capacitor | Constant, represents the pacing PWM output that controls the amplitude of the pace |
| PACE_CHARGE_CTRL_D2 | 1 | Starts and stops PACING_REF_PWM_D5 (charging capacitor) | High in charging, low in pacing. This prevents the capacitor from charging while it's emitting the charge as a pulse to the atrium. |
| PACE_GND_CTRL_D10 | 0 | Controls current flow to the tip of the atrium from the ring | Low in charging, high in pacing. Prevents pulse from delivering too much charge to the atrium. |
| ATR_PACE_CTRL_D8 | 0 | Discharges primary capacitor through atrium | Low in charging, high in pacing state. Indicates when capacitor releases charge |
| ATR_GND_CTRL_D11 | 1 | Stops charge buildup during primary capacitor discharge | Capacitor has a charge limit. |
| Z_ATR_CTRL_D4 | 0 | Analyzes impedance at atrial electrode and connection between the atrial electrodes/atrium | Always kept to low to prevent circuit from short circuiting. |

| | | | |
|---|---|---|---|
| LED_STATE_RED<br>LED_STATE_BLUE<br>LED_STATE_GREEN | 1<br>0<br>0 | Indicates change in charging and pacing state | *Note: Implemented for demonstration purposes only |
| Pulse_Period | 60000 [msec/min] / 60 [BPM]<br><br>= 1000 [msec/beats] | Converts BPM to total time from the end of one pulse to the end of another pulse in msec | Responsible for transition timing between charging and pacing state. |
| Pulse_Width | 20 [msec] | Length of the pulse, discharging duration of capacitor in msec | Measure of time between single pulse of energy. |

*Table 5: AOO Variable Descriptions*

**VVI:**

| **Variables:** | **Initialization:** | **Function:** | **Description:** |
|---|---|---|---|
| **Monitor:** | | | |
| VENT_CMP_DETECT_D1 | 0 | Ventricular Sensing | |
| PushButton_Inhibit | 0 | Button pushed indicates ‘true’ to inhibit pulse, button not pushed indicates ‘false’ and pace is not inhibited | |
| **Control:** | | | |
| PACING_REF_PWM_D5 | P_Amp<br><br>= 100 [%] | Charges primary capacitor | Constant, represents the pacing PWM output that controls the amplitude of the pace |
| PACE_CHARGE_CTRL_D2 | 1 | Starts and Stops the PACING_REF_PWM_D5 control (charging of capacitor) | Can’t be high if VENT_PACE_CTRL_D9 is also high or the patient’s ventricle could be connected directly to the PWM signal (safety hazard) |
| VENT_PACE_CTRL_D9 | 0 | Discharges the primary capacitor through the ventricle | Cannot be set to high since patient’s ventricle will be connected directly to PWM signal |
| VENT_GND_CTRL_D12 | 1 | Stops charge buildup during primary capacitor discharge | High in charging, low in pacing state. |

| | | | |
|---|---|---|---|
| | | | Prevents capacitor from overcharging. |
| PACE_GND_CTRL_D10 | 0 | Controls current flow to tip of ventricle from the ring | Must be set to high as it controls tip switch |
| Z_VENT_CTRL_D7 | 0 | Connects the impedance circuit and the ventricle ring electrode | Always kept to low to prevent circuit from short circuiting. |
| FRONTEND_CTRL_D13 | 1 | Begins to sense circuitry within model | |
| VENT_CMP_REF_PWM_D3 | S_Amp<br><br>= 80 [%] | Establishes threshold for ventricular action to notify when sensing occurs | Constant value which represents sensing PWM threshold in sensing circuit |
| Pulse_Period | 60000 [msec/min] / 60 [BPM]<br><br>= 1000 [msec/beats] | Converts BPM to total time from the end of one pulse to the end of another pulse in msec | Responsible for transition timing between charging and pacing state. |
| Pulse_Width | 20 [msec] | Length of the pulse, discharging duration of capacitor in msec | Measure of time between single pulse of energy. |
| VRP | 100 | Constant, represents time interval (msec) after a ventricular event, when ventricular sensing will not inhibit or trigger. | |
| LED_STATE_RED<br>LED_STATE_BLUE<br>LED_STATE_GREEN | 0<br>1<br>0 | Indicates change in charging and pacing state | *Note: Implemented for demonstration purposes only |

*Table 6: VVI Variable Descriptions*

**AAI:**

| **Variable:** | **Initialization:** | **Function:** | **Description:** |
|---|---|---|---|
| **Monitor:** | | | |
| ATR_CMP_DETECT_D0 | 0 | Atrium Sensing | |
| PushButton_Inhibit | 0 | Pushed outputs declared true to inhibit pulse, not pushed outputs | |

| | | | |
|---|---|---|---|
| | | declared false and pace isn't inhibited | |
| **Control:** | | | |
| PACING_REF_PWM_D5 | P_Amp | Charges Primary Capacitor | Constant, represents the pacing PWM output that controls the amplitude of the pace |
| PACE_CHARGE_CTRL_D2 | 1 | Starts and Stops PACING_REF_PWM_D5 (charging of capacitor) | Can't be high if VENT_PACE_CTRL_D8 is also high or the patient's atrium could be connected directly to the PWM signal |
| ATR_PACE_CTRL_D8 | 0 | Discharges the primary capacitor through the atrium | Can't be set to high or the patient's atrium could be connected directly to the PWM signal |
| ATR_GND_CTRL_D11 | 1 | Stops charge buildup during primary capacitor dsicharge | Capacitor has charge limit, high in chargin and low in pacing state |
| PACE_GND_CTRL_D10 | 0 | Controls current flow to tip of ventricle from the ring | Has to be high as it controls the switch which follows the tip |
| Z_ATR_CTRL_D7 | 0 | Controls and analyzes impedance at the atrial electrode, and the connection between the atrial electrodes and the atrium | Always set to low to keep the circuit closed. Prevents it from short circuiting. |
| FRONTEND_CTRL_D13 | 1 | Starts sensing circuitry | |
| ATR_CMP_REF_PWM_D3 | S_Amp | Limit for when atrial action potential should be sensed | Constant, represents the sensing PWM threshold in the sensing circuit |
| Pulse_Period | 60000 [msec/min] / 60 [BPM]<br><br>= 1000 [msec/beats] | Converts BPM to total time from the end of one pulse to the end of another pulse in msec | Responsible for transition timing between charging and pacing state. |

| | | | |
|---|---|---|---|
| Pulse_Width | 20 [msec] | Length of the pulse, discharging duration of capacitor in msec | Measure of time between single pulse of energy |
| ARP | 150 | Constant, represents the programmed time interval following an atrial event where they will not inhibit or trigger pacing. The Atrial Refractory Period is measured in msec | |
| LED_STATE_RED<br>LED_STATE_BLUE<br>LED_STATE_GREEN | 1<br>0<br>0 | Indicates change in charging and pacing state | *Note: Implemented for demonstration purposes only |

*Table 7: AAI Variable Descriptions*

## State flow Conditional Decision

| Conditional | Used Modes | Description |
|---|---|---|
| after(Pulse_Period – Pulse_Width, msec) | VOO, AOO | Defines transition timing between charging and pacing states (in milliseconds) |
| after(Pulse_Width, msec) | VOO, AOO, VVI, AAI | How long the capacitor discharges for; once pacing finishes, the system returns back to charging after Pulse Width (in milliseconds) |
| after(VRP-Pulse_Width, msec) | VVI | Creates a time delay between the charging and pacing states based on the Ventricular Refractory Period and the width of the previous pulse (in milliseconds) |
| after(ARP-Pulse_Width, msec) | AAI | Creates a time delay between the charging and pacing states based on the Atrial Refractory Period and the width of the previous pulse (in milliseconds) |

| | | |
|---|---|---|
| after(Pulse_Period-VRP, msec)[VENT_CMP_DETECT_D0 == 0 & PushButton_Inhibit == false] | VVI | Will only move to the next state (pacing) if it remains 0 and inhibit push button is not held down |
| after(Pulse_Period-ARP, msec)[ATR_CMP_DETECT_D0 == 0 & PushButton_Inhibit == false] | AAI | Will only move to the next state (pacing) if it remains 0 and inhibit push button is not held down |
| [VENT_CMP_DETECT_D0 == 1 \| PushButton_Inhibit == true] | VVI | Will move back to the last state (charging) if it is 1 or inhibit push button is held down |
| [ATR_CMP_DETECT_D0 == 1 \| PushButton_Inhibit == true] | AAI | Will move back to the last state (charging) if it is 1 or inhibit push button is held down |
| after(Pulse_Period,msec)[PushButton_Change_States == true] | Reset_VOO_to_VVI, Reset_VVI_to_AOO, Reset_AOO_to_AAI, Reset_AAI_to_VOO | Will allow for modes to properly reset if change states push button is held down<br><br>*Note: only for demonstration purposes |

*Table 8: Stateflow conditional descriptions*

## State Transition Table

**VOO:**

| **Current State** | **Variables During State Change** | | | **Result** |
|---|---|---|---|---|
| Charging (Start State) | **Variable** | **Current State Value** | **Next State Value** | Capacitor is charged |
| | PACE_CHARGE_CTRL_D2 | 1 | 0 | |
| | VENT_PACE_CTRL_D9 | 0 | 1 | |
| | VENT_GND_CTRL_D12 | 1 | 0 | |
| | PACE_GND_CTRL_D10 | 0 | 1 | |
| | Z_VENT_CTRL_D7 | 0 | 0 | |
| | LED_STATE_RED | false | true | |
| | LED_STATE_BLUE | true | false | |
| | LED_STATE_GREEN | false | false | |
| Pacing | **Variable** | **Current State Value** | **Next State Value** | Pulse is emitted |

| | Variable | Current State Value | Next State Value | |
|---|---|---|---|---|
| | PACE_CHARGE_CTRL_D2 | 0 | 1 | |
| | VENT_PACE_CTRL_D9 | 1 | 0 | |
| | VENT_GND_CTRL_D12 | 0 | 1 | |
| | PACE_GND_CTRL_D10 | 1 | 0 | |
| | Z_VENT_CTRL_D7 | 0 | 0 | |
| | LED_STATE_RED | false | true | |
| | LED_STATE_BLUE | true | false | |
| | LED_STATE_GREEN | false | false | |

*Table 9: VOO State Transition Table*

**AOO:**

| **Current State** | **Variables During State Change** | | | **Result** |
|---|---|---|---|---|
| Charging (Start State) | **Variable** | **Current State Value** | **Next State Value** | Capacitor is charged |
| | PACE_CHARGE_CTRL_D2 | 1 | 0 | |
| | ATR_PACE_CTRL_D8 | 0 | 1 | |
| | ATR_GND_CTRL_D11 | 1 | 0 | |
| | PACE_GND_CTRL_D10 | 0 | 1 | |
| | Z_ATR_CTRL_D4 | 0 | 0 | |
| | LED_STATE_RED | true | false | |
| | LED_STATE_GREEN | false | true | |
| | LED_STATE_BLUE | false | false | |
| Pacing | **Variable** | **Current State Value** | **Next State Value** | Pulse is emitted |
| | PACE_CHARGE_CTRL_D2 | 0 | 1 | |
| | ATR_PACE_CTRL_D8 | 1 | 0 | |
| | ATR_GND_CTRL_D11 | 0 | 1 | |
| | PACE_GND_CTRL_D10 | 1 | 0 | |
| | Z_ATR_CTRL_D4 | 0 | 0 | |
| | LED_STATE_RED | false | true | |
| | LED_STATE_GREEN | true | false | |
| | LED_STATE_BLUE | false | false | |

*Table 10: AOO State Transition Table*

**VVI:**

| **Current State** | **Variables During State Change** | **Result** |
|---|---|---|

| | | | | |
|---|---|---|---|---|
| Charging (Start State) | **Variable** | **Current State Value** | **Next State Value** | Charging the capacitor |
| | PACE_CHARGE_CTRL_D2 | 1 | 0 | |
| | VENT_PACE_CTRL_D9 | 0 | 1 | |
| | VENT_GND_CTRL_D12 | 1 | 0 | |
| | PACE_GND_CTRL_D10 | 0 | 1 | |
| | FRONTEND_CTRL_D13 | 0 | 0 | |
| | LED_STATE_RED | false | true | |
| | LED_STATE_BLUE | true | false | |
| | LED_STATE_GREEN | false | false | |
| | VENT_CMP_DETECT_D0 | Only goes to pacing state if its still false after Pulse_Period – VRP amount of time (msec) | | |
| | PushButton_Inhibit | Only goes to pacing state if its still false after Pulse_Period – VRP amount of time (msec) | | |
| Pacing | **Variable** | **Current State Value** | **Next State Value** | Sending the pulse for pacing |
| | PACE_CHARGE_CTRL_D2 | 0 | 1 | |
| | ATR_PACE_CTRL_D9 | 1 | 0 | |
| | ATR_GND_CTRL_D12 | 0 | 1 | |
| | PACE_GND_CTRL_D10 | 1 | 0 | |
| | Z_VENT_CTRL_D13 | 0 | 0 | |
| | LED_STATE_RED | true | false | |
| | LED_STATE_BLUE | false | true | |
| | LED_STATE_GREEN | false | false | |

*Table 11: VVI State Transition Table*

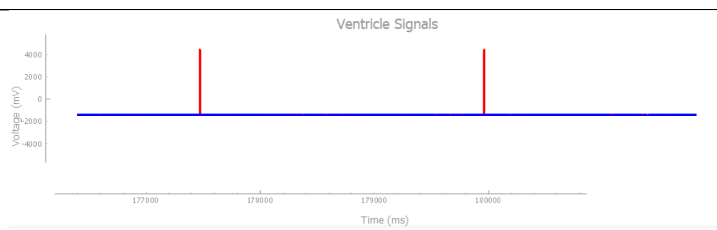
**AAI:**

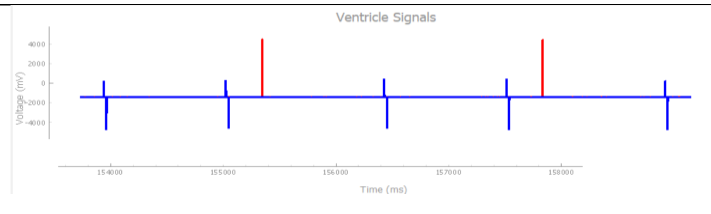
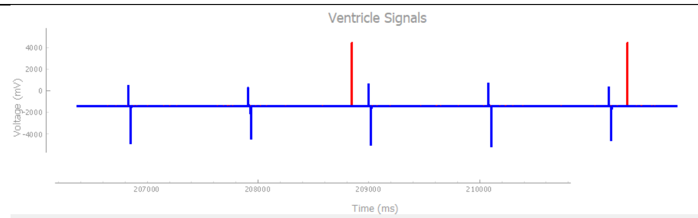
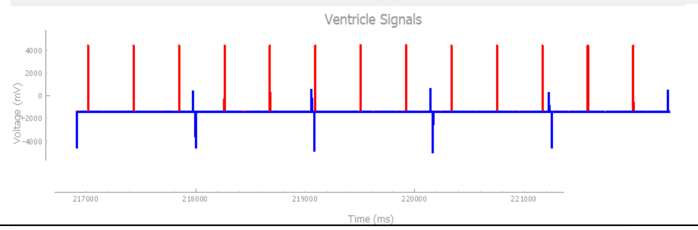
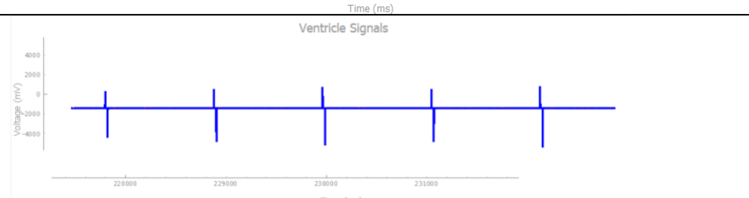
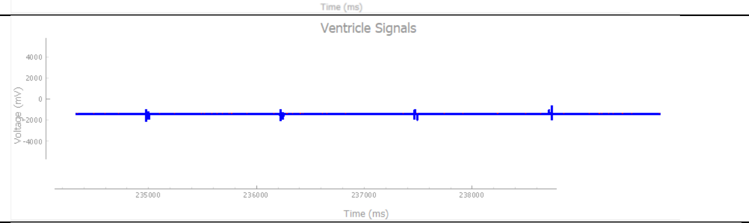
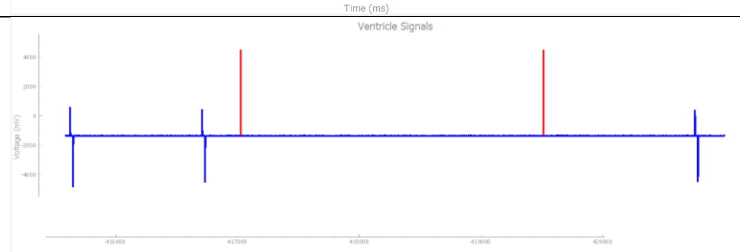
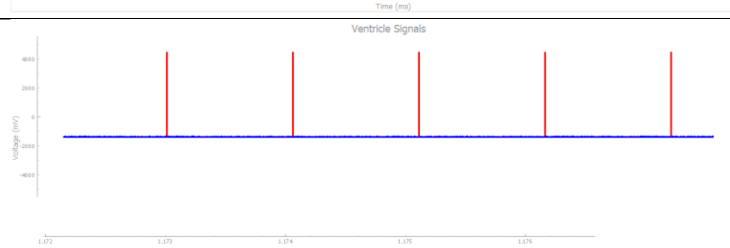
| **Current State** | **Variables During State Change** | | | **Result** |
|---|---|---|---|---|
| Charging (Start State) | **Variable** | **Current State Value** | **Next State Value** | Charging the capacitor |
| | PACE_CHARGE_CTRL_D2 | 1 | 0 | |

| | | | | |
|---|---|---|---|---|
| | ATR_PACE_CTRL_D8 | 0 | 1 | |
| | ATR_GND_CTRL_D11 | 1 | 0 | |
| | PACE_GND_CTRL_D10 | 0 | 1 | |
| | Z_ATR_CTRL_D4 | 0 | 0 | |
| | LED_STATE_RED | true | false | |
| | LED_STATE_GREEN | false | true | |
| | LED_STATE_BLUE | false | false | |
| | ATR_CMP_DETECT_D0 | Only goes to pacing state if its still false after Pulse_Period – ARP amount of time (msec) | | |
| | PushButton_Inhibit | Only goes to pacing state if its still false after Pulse_Period – ARP amount of time (msec) | | |
| Pacing | **Variable** | **Current State Value** | **Next State Value** | Sending the pulse for pacing |
| | PACE_CHARGE_CTRL_D2 | 0 | 1 | |
| | ATR_PACE_CTRL_D8 | 1 | 0 | |
| | ATR_GND_CTRL_D11 | 0 | 1 | |
| | PACE_GND_CTRL_D10 | 1 | 0 | |
| | Z_ATR_CTRL_D4 | 0 | 0 | |
| | LED_STATE_RED | false | true | |
| | LED_STATE_GREEN | true | false | |
| | LED_STATE_BLUE | false | false | |

*Table 12: AAI State Transition Table*

## Test Cases

| Test Case | Expected Outcome | Actual Outcome | Pass/Fail |
|---|---|---|---|
| S_Amp = 40 | Sensing threshold too low, will not register electrical signal |  | Pass |

| | | | |
|---|---|---|---|
| S_Amp = 80 | Sensing threshold 'normal', will register electrical signal |  | Pass |
| S_Amp = 100 | Sensing threshold too high, will send paces no matter what the pulse is |   | Pass |
| P_Amp = 100 | Pace amplitude normal size (100%) |  | Pass |
| P_Amp = 50 | Pace amplitude half the size (50%) |  | Pass |
| Quickly push Inhibit Button instead of holding it | Inhibits pace only briefly rather than for extended duration |  | Pass |
| BPM greater than 60 in Heartview | Will inhibit pulses |  | Pass |

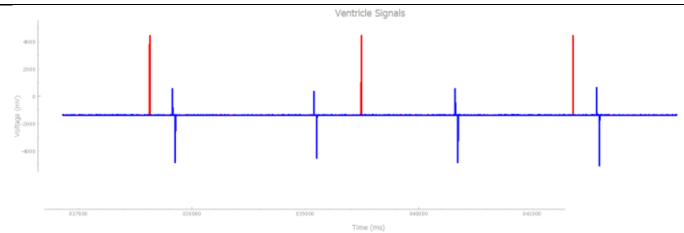
| | | | |
|---|---|---|---|
| BPM less than 60 in Heartview | Will continue to deliver pulses |  | Pass |

*Table 13: Simulink and HeartView Test Cases*

# Device Controller-Monitor (DCM)

The device controller-monitor functions as an interface for healthcare professionals to communicate with the pacemaker device. The DCM has the capability to configure a multitude of parameters within the pacemaker, tailored to the unique requirements of individual patients. Additionally, it possesses the ability to differentiate between distinct patients and maintain a comprehensive user history database. The DCM is equipped to present diagnostic information, trends, and specific pacemaker data through various visualization techniques, enabling the doctor to gain a comprehensive understanding of the device's performance, and the patient diagnosis.

## Requirements

The following requirements for the DCM are listed below. Each requirement is provided an ID to be traceable to the design and vice versa.

| **DCM Requirements** | |
|---|---|
| **ID** | **Requirement** |
| DCM_1 | Develop a user interface that can manage windows for displaying text and graphics. |
| DCM_2 | Develop a user interface that can process user position and input buttons. |
| DCM_3 | Develop a starting interface which prompts users to login as an existing user or register with a new name and password. |
| DCM_4 | Develop a storage system that provides permanent storage of user data. |
| DCM_5 | Allow a maximum of 10 users to be registered, with an option to delete existing users. |
| DCM_6 | Develop a pacemaker interface to present the following pacing modes for review: AOO, VOO, AAI, VVI. |

| | |
|---|---|
| DCM_7 | Develop a pacemaker interface to store programmable parameters for each pacing mode and allow the user to modify them as they like. |
| DCM_8 | Provide an option for the user to connect/disconnect the DCM and the pacemaker device, a status of its connection, and if a new pacemaker device has been connected. |
| DCM_9 | Provide a visualization of the egram data in real-time from the pacemaker device (only make frontend at this time). |

*Table 14: DCM Requirements*

## Planning

The initial phase involves the selection of programming languages for configuring the DCM. A multitude of programming languages are available to develop a user interface. However, each of these languages presents its own set of advantages and drawbacks. In light of this, our approach is to use a weighted decision matrix. This matrix assigns specific weights to individual criteria, considering their significance in enhancing the DCM's functionality. Each programming language is evaluated based on its effectiveness in delivering on these criteria, as well as the feasibility for developers to work with it.

| **DCM Decision Matrix** | | | | |
|---|---|---|---|---|
| **Criteria and weight** | **Python** | **Node.js/React.js** | **C/C++** | **Java** |
| **Aesthetic - 2** | 3 | 5 | 1 | 2 |
| **User Interface - 4** | 5 | 5 | 3 | 3 |
| **Communication - 4** | 5 | 2 | 3 | 3 |
| **User Storage - 3** | 5 | 5 | 3 | 5 |
| **Total / 65** | **61** | 53 | 35 | 43 |

*Table 15: Programming Language Decision Matrix*

As per the outcomes of the decision matrix, Python emerges as the optimal programming language choice. Python, with its Tkinter library, is well-suited for crafting a user-friendly interface. Moreover, Python offers a serial communication library for robust device communication. In terms of user data storage, Python's compatibility with JSON files ensures the seamless management of persistent memory.

Python excels in each criterion, demonstrating strong performance and offering a developer-friendly environment. The sole limitation is in the aesthetic aspect, given that Tkinter provides a more traditional design. However, it is reasonable to mitigate this drawback as it does not compromise the functionality or efficiency of the Device Controller-Monitor (DCM).

## Design

The DCM and its functionalities can be separated into distinct modules, facilitating a design approach that emphasizes modularity, simplicity, organization, and reusability of various functions and components. Each module is designed to be implemented separately while maintaining the capability to interface with other modules for data exchange and the augmentation of functionalities. Within the diagram, each module is labelled with its title and primary functions. Arrows in the diagram serve to indicate communication between modules, illustrating data flow and function dependencies. A flow chart was also designed to display all possible states of the DCM design.

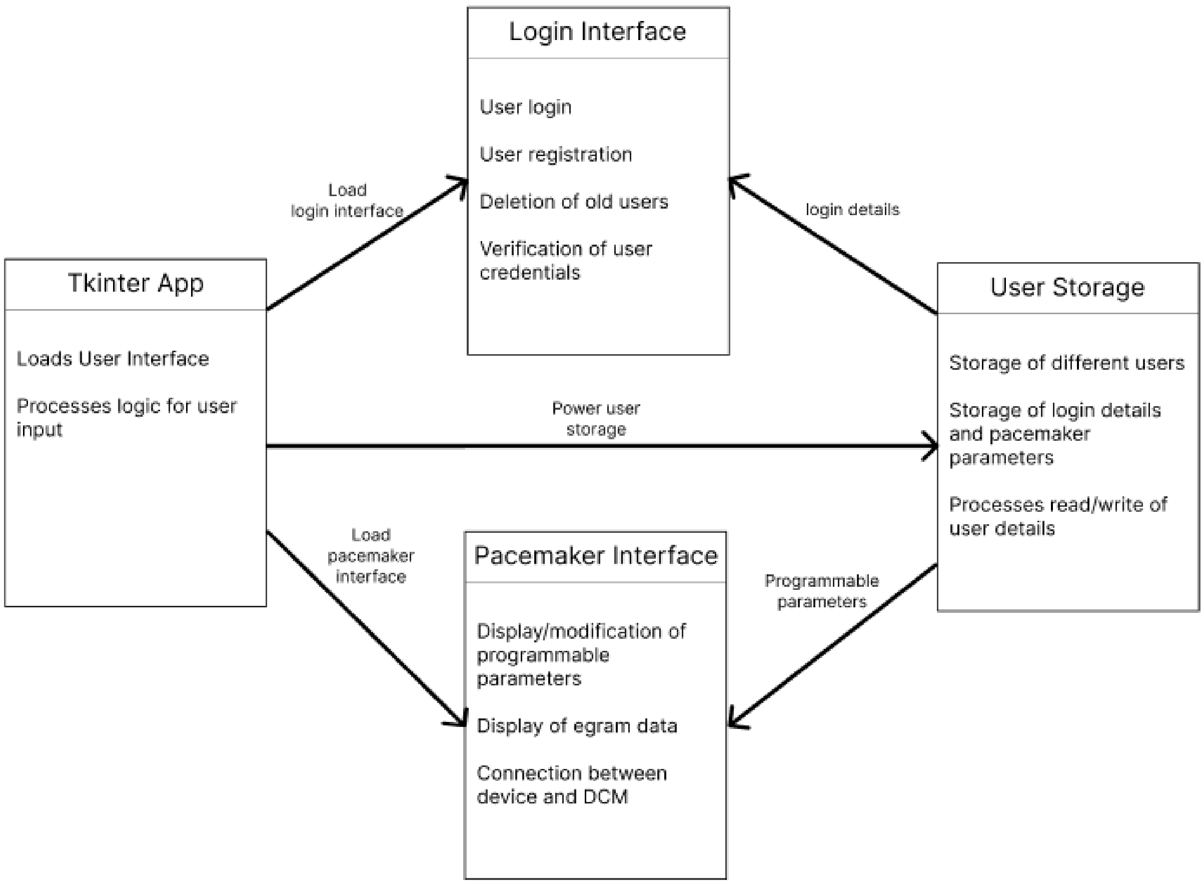



*Figure 13: DCM*

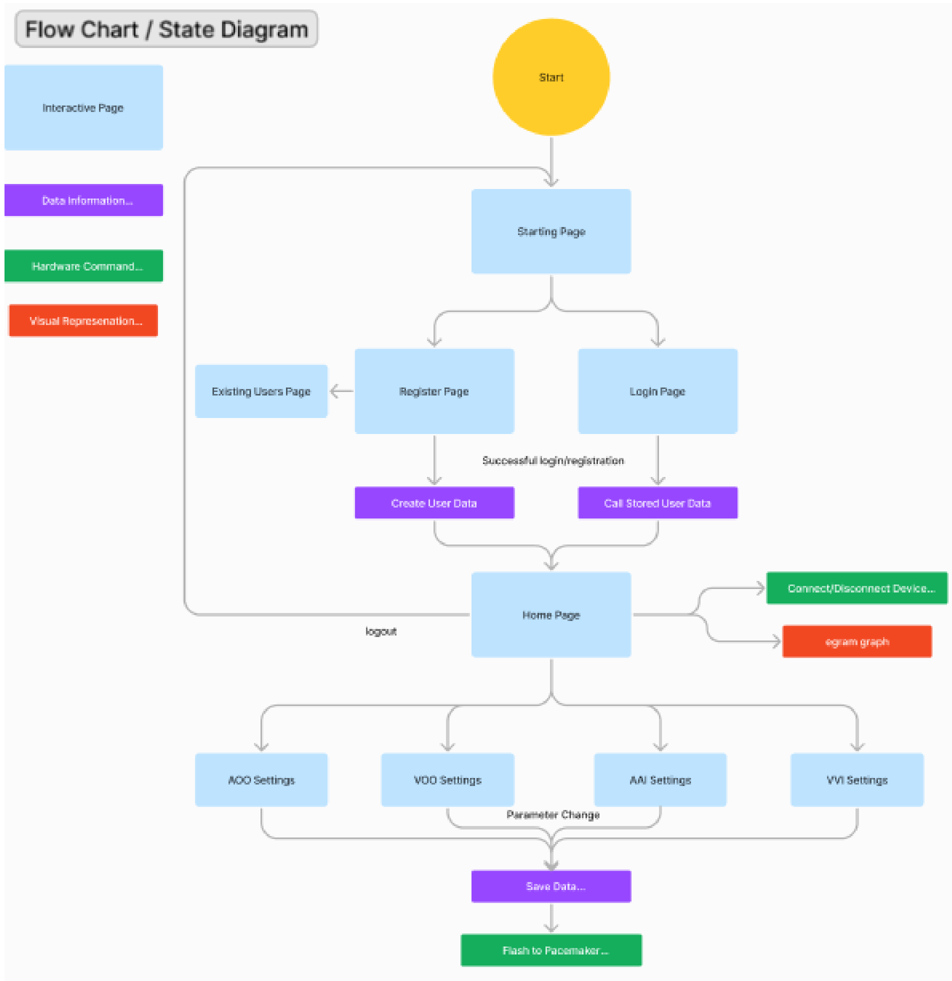



*Figure 14: State flow Chart*

## Modules

Each module is described with their functionality, their requirements, what DCM requirement it traces back to, details on the public and private functions it has access to, and state variables they work with.

<table>
<tr><th colspan="2">Tkinter App Module</th></tr>
<tr><th>Purpose</th><th>Secrets</th></tr>
<tr><td>This module is responsible for configuring the GUI and creating each component of the interface when the program is activated. It provides user interface, logic for user input, and starting protocols to other modules.</td><td><ul><li>How different interfaces are loaded.</li><li>How components of the interface are programmed.</li><li>How logic is processed for user input and output.</li></ul></td></tr>
<tr><th>Module Specific Requirement</th><th>Requirement it fulfills</th></tr>
</table>

| | |
|---|---|
| The app shall display different interfaces/pages, each page serving a different function for the user. | DCM_1 |
| The app shall display different titles, text, and descriptions for each interface/page, describing its purpose and possible user inputs. | DCM_1 |
| The app shall provide user input components such as forms and buttons. It will be able to process these inputs into outputs. | DCM_2 |
| The app shall activate the user storage when the program is activated, so it is ready for read/write protocols. | DCM_4 |

**Global Variables**

| **Variable name** | **Details** |
|---|---|
| box | Box is the frame that holds all the components of the GUI, such as new pages, text, buttons, and more. New components are appended to it or cleared. Box is a data structure object that takes input for configuration of the GUI, such as title and dimensions. |

**Public Functions**

| **Function and parameters** | **Details** |
|---|---|
| mainloop() – No parameters | Starts the GUI to be constantly run until exited. Sets up the box object with dimensions and title for the GUI. |
| protocol(name,func) | Name holds the name of an event that can occur by the GUI. Func holds the name of the function to be called when that event occurs. Protocol sets an event listener to connect the two variables. |
| redirectPage() – No parameters | Deletes all components on current interface, wiping it clean. Returns a new blank interface object to be modified by the function that called redirectPage. |
| Label(parent,text,font,padx,pady) | Displays the text variable string on the frame given from the parent variable. Font variable sets font size of the text. Padx and pady set the padding of the text, placing it in the correct position. |
| Button(parent,text,font,command,padx,pady) | Displays a button on the frame given by parent, displaying the string by the text variable. Font variable sets font size, padx and pady set the padding of the button. Command sets the function to be called when |

| | |
|---|---|
| | the button is clicked, such as redirecting to a new page. |
| Entry(parent) | Displays an entry form on the frame given by the parent, where users can write text into it. |
| showInfo(title, text, parent) | Displays a pop-up window with a message (“text”) in the parent frame. Mostly used for confirmation or information being shared to the user when a command is executed. Title is the header of the window. |
| askquestion(title, text) | Opens a pop-up window with a message (“text”) and title (“title”) in the parent frame, with an option to click “yes” or “no”. Returns True if “yes” has been clicked, and False if “no” has been clicked. |
| Scale(parent,from_,to_,length, orient,value,command) | Displays a slider bar on the parent frame with range [from_to,to_]. The length and orient of the slider is given by the length and orient variable. The value variable sets the initial value of the slider. Command sets the function to be called when the slider is modified, so a set of instructions can be used for when a value is modified. |

*Table 16: Tkinter App Module*

| **User Storage Module** | |
|---|---|
| **Purpose** | **Secrets** |
| This module is responsible for permanently storage, review, and modification of programmable parameters and login details for each user. | • How user details are stored permanently.<br>• How user details are read and written. |

| **Module Specific Requirement** | **Requirement it fulfills** |
|---|---|
| The user storage shall permanently store user login details and programmable parameters for the pacemaker. | DCM_4<br>DCM_7 |
| The user storage shall store details for up to 10 users. | DCM_5 |
| The user storage shall have read/write protocols for review and modification of user details. | DCM_3<br>DCM_7 |

| | |
|---|---|
| | |
| **Global Variables** | |
| **Variable name** | **Details** |
| userData | Holds all the current user data from when the GUI is activated, saving every change to write to the user storage when it is deactivated. Structure is a dictionary that holds an array of values for each user. The array holds the username, the password, and the values for each pacing mode. |
| **Public Functions** | |
| **Function and parameters** | **Details** |
| loadUserData() – No parameters | Loads the userData.json file in read mode and copies its data to the userData variable, for all other modules to read/write old and new data. If the userData.json file does not exist, a new one is created and userData is initialized as empty. |
| saveUserData() – No parameters | Loads the userData.json file in write mode and fills the file with the current object userData is holding. |

*Table 17: User Storage Module*

| **Login Interface Module** | |
|---|---|
| **Purpose** | **Secrets** |
| This module is responsible for welcome the user with a login/registration interface that allows them to login with an existing user or create a new one. | • How login details are read and stored.<br>• How login details are verified.<br>• How a new account is registered.<br>• How past users are deleted. |
| **Module Specific Requirement** | **Requirement it fulfills** |
| The interface shall allow the user to create a new user account on the registration page, with a username and a password. | DCM_3 |

| | |
|---|---|
| The interface shall allow a user to login on the login page with an existing user account's correct details, redirecting them to the pacemaker interface. | DCM_3 |
| The interface shall notify the user when they have inputted wrong/forbidden user details during login or registration. Forbidden details consist of an empty username/password, or an already existing username. | DCM_3 |
| The interface shall allow the user to view the details of existing users with an option to delete them, on the existing users' page. | DCM_5 |
| **Global Variables** | |
| **Variable name** | **Details** |
| currentUser | Holds the user info of the current user logged in. |
| **Public Functions** | |
| **Function and parameters** | **Details** |
| startPage() – No parameters | Loads the starting page of the login interface, where the user is prompted to login or register. A title, description, and two buttons are provided. When the button labelled "Login" is clicked, it calls the loginPage() function, redirecting the user to the login page. The second button does the same but is labelled "Register" and calls registerPage(). |
| loginPage() – No parameters | Loads the login page. The function sets up two forms for username and password, a login button that calls authenticateUser(), and a back button that calls startPage(). |
| registerPage() – No parameters | Loads the register page. The function sets up two forms for username and password, a register button that calls registerUser(), an existing user's button that calls existingUsersPage(), and a back button that calls startPage(). |
| existingUsersPage() – No parameters | Loads the existing user's page. The function provides a description of how many users have been registered. It also provides a button for each user that calls deleteUser(username). |
| **Private Functions** | |
| **Function and parameters** | **Details** |

| | |
|---|---|
| authenticateUser() – No parameters | Retrieves the username and password from the two forms. Provides a specific error message if either or both forms are empty, if the password is incorrect, or if the username does not exist. If credentials are correct, sets the currentUser variable to the data of the user logged in, and calls the homePage() function, redirecting the user to the pacemaker interface. |
| registerUser() – No parameters | Retrieves the username and password from the two forms. Provides a specific error message if either or both forms are empty, if there are already 10 users registered, or if the username already exists. If there is no error, the newUser object is set up, appended to the storage, the currentUser variable is set to newUser, and homePage() is called to redirect the user to the pacemaker interface. |
| deleteUser(username) | Searches the user storage and deletes the object that has the same username as the inputted username variable. Provides a message box informing the user that the deletion was successful, and calls existingUsersPage() to reload the page, removing the old user from the interface. |

*Table 18: Login Interface Module*

| **Pacemaker Interface Module** | |
|---|---|
| **Purpose** | **Secrets** |
| This module is responsible for providing the user with the ability to review/modify pacemaker details. This includes all pacing modes and their parameters, egram data, and the device connection status. | • How the user specific parameters are retrieved.<br>• How programmable parameters are displayed and saved when modified.<br>• How a unique pacemaker device is detected.<br>• How the device connection process works. |
| **Module Specific Requirement** | **Requirement ID it fulfills** |
| The interface shall allow the user to review the programmable parameters of each pacing mode in their respective page. | DCM_6 |
| The interface shall allow the user to save the programmable parameters that they have modified for the specific pacing mode. | DCM_7 |

| | |
|---|---|
| The interface shall enforce the acceptable ranges for each programmable parameter during modification, not allowing any other value. | DCM_7 |
| The interface shall allow the user to connect/disconnect the pacemaker device to the DCM with a button. The button title will describe what will be done if the user clicks the button. | DCM_8 |
| The interface shall allow the user to redirect to an egram data page when the egram button is clicked. | DCM_9 |
| The interface shall allow the user to logout of the current user account with the logout button, redirecting them to the login interface. | DCM_3 |

**Global Variables**

| **Variable name** | **Details** |
|---|---|
| currentUser | Holds the user info of the current user logged in. |

**Public Functions**

| **Function and parameters** | **Details** |
|---|---|
| homePage() – No parameters | Loads the home page. Sets up a button for each pacing mode that calls settingsPage with the mode as a parameter. Sets up a connect button that calls connectDevice(). Sets up an egram button that calls egramPage(). Sets up a logout button that calls startPage(). |
| settingsPage(mode) | Loads the settingsPage of the mode provided. Sets up sliders for each programmable parameter of that pacing mode that calls the respective change function for whenever a modification is made to a parameter. Sets up a save button that calls saveData(). Sets up a flash button that calls flashCode(). Sets up a back button that calls homePage(). |
| egramPage() – No parameters | Loads the egram page. The function sets up a back button that calls homePage(). |

**Private Functions**

| **Function and parameters** | **Details** |
|---|---|

| | |
|---|---|
| connectDevice() – No parameters | Connects the pacemaker device and switches text to "disconnect" if current text on button is "connect". Disconnects the pacemaker device and switches text to "connect" if current text is "disconnect". Notifies the user with a message box if the connected device is a new device. |
| saveData() – No parameters | Saves the data of the current mode being modified to the currentUser variable. |
| flashCode() – No parameters | Notifies the user with a message box that the user data has been flashed to the pacemaker. |
| LRLHChange(value, LorH) | Called when LRL or HYST has been modified. LorH is provided to know which value has been modified. Changes value variable to new value by user. Updates text with new value on interface to display to user. These variables have special increments, thus are not used in the generalChange function. Overall, prevents a non-eligible value from being chosen on the slider. |
| VAPWChange(value, VorA) | Called when VPW or APW has been modified. VorA is provided to know which value has been modified. Changes value variable to new value by user. Updates text with new value on interface to display to user. These variables have special increments, thus are not used in the generalChange function. Overall, prevents a non-eligible value from being chosen on the slider. |
| AVSensChange(value, VorA) | Called when AS or VS has been modified. VorA is provided to know which value has been modified. Changes value variable to new value by user. Updates text with new value on interface to display to user. These variables have special increments, thus are not used in the generalChange function. Overall, prevents a non-eligible value from being chosen on the slider. |
| VAAmpChange(value, VorA) | Called when AA or VA has been modified. VorA is provided to know which value has been modified. Changes value variable to new value by user. Updates text with new value on interface to display to user. These variables have special increments, thus are not used in the generalChange function. Overall, prevents a non-eligible value from being chosen on the slider. |
| generalChange(value, increment, type, unit) | Called when URL, ARP, VRP. PVARP, or RS has been modified. Type is provided to know which value has been modified. Changes value variable to new value by user. Increment is provided for function to determine how much to increment slider when updated. Updates text with new value and unit on interface to display to user. Overall, prevents a non-eligible value from being chosen on the slider based on the increments. |

*Table 19: Pacemaker Interface Module*

## Design Decisions and Likely Changes

Over the course of our DCM design process, we have undertaken a multitude of design choices. Presented below are these design decisions alongside their justification. It is important to note that a number of these decisions may undergo revisions in the future, and consequently, we have provided additional insights on the reason why they may be changed.

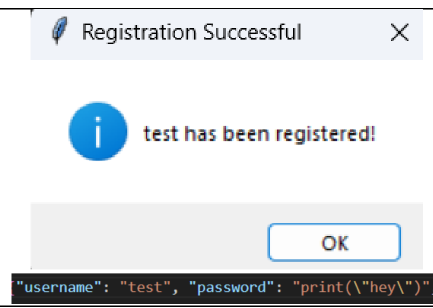
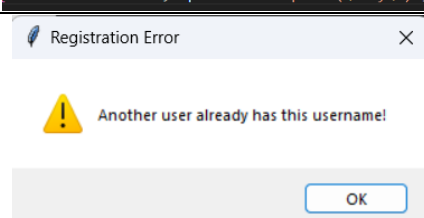
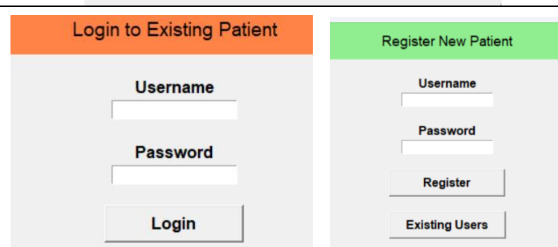
<table>
<tr><th colspan="6">Design decisions made</th></tr>
<tr><th colspan="3">Design Decision</th><th colspan="3">Details</th></tr>
<tr><td colspan="3">Show all existing users and provide the option to delete them from the list.</td><td colspan="3">To make it easy to optimize the users when you reach the 10 users max we wanted to provide the option to chose a user on the account to delete.</td></tr>
<tr><td colspan="3">Register page goes automatically to the home page.</td><td colspan="3">We feel that it is ambiguous to have to go back and log in after registering so we added this action.</td></tr>
<tr><td colspan="3">All text at the top of the pages</td><td colspan="3">We believe that a good GUI requires user satisfaction, so we aimed to make the interface as friendly as possible.</td></tr>
<tr><td colspan="3">Use of buttons and not having pop-out pages.</td><td colspan="3">We made sure not to have pop out pages and only have pop-outs for messages so that the users screen does not feel clustered</td></tr>
<tr><td colspan="3">Easy navigation through pages</td><td colspan="3">Use of detailed buttons like “back” or “logout” which make it easy to navigate between pages and for the user to use the interface.</td></tr>
<tr><th colspan="6">Design decisions likely to be changed</th></tr>
<tr><th colspan="2">Design Decision</th><th colspan="2">Details</th><th colspan="2">Why it is likely to change</th></tr>
<tr><td colspan="2">Putting the connect to device and develop egram data on the home page</td><td colspan="2">We did this because we believe it is easiest to have this on the general page at the moment without the actual serial inputs yet.</td><td colspan="2">Once we begin to program the serial requirements for the next assignment this may need to change based on how the code works.</td></tr>
<tr><td colspan="2">Use only one main.py file for our GUI</td><td colspan="2">We decided to do this because instead of having a mass amount of files to separate between pages we believed that one file with all of the code would be sufficient</td><td colspan="2">May need to change once more requirements are implemented and serial is added because we are already at 500 lines of code.</td></tr>
<tr><td colspan="2">Using functions for all of the pages and commands</td><td colspan="2">We used functions instead of object oriented programming because we believed it would be</td><td colspan="2">We may have to switch to an object-oriented programming model to separate the</td></tr>
</table>

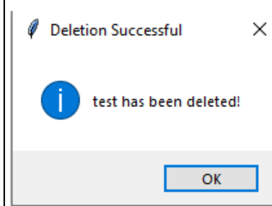
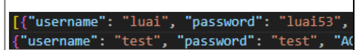
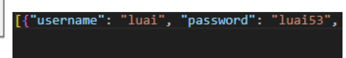
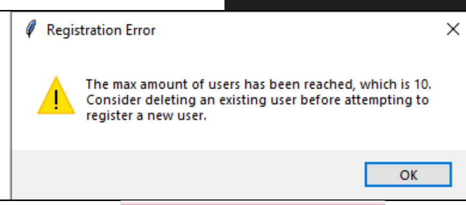
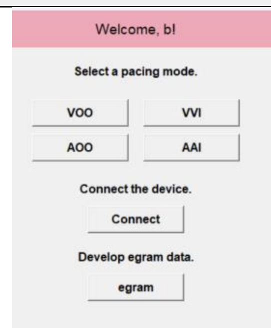
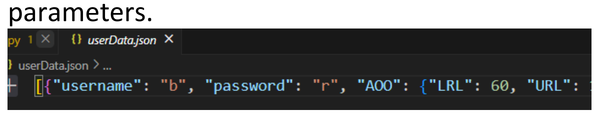
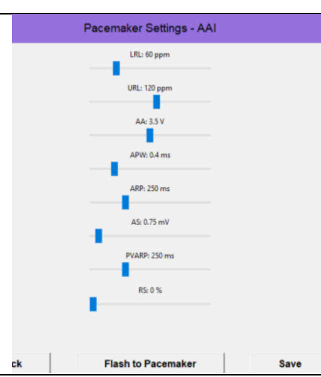
| | | |
|---|---|---|
| | a bit overkill for the current requirements and may have been more code then the functions made. | functionality of each page and module. |
| Have sliders for editing pacing settings as well as displaying the current value below the slider. Sliders only shift to correct values. | This makes it extremely easy to change your pacing settings and not have to type values and potentially enter an incorrect value. | Our use of sliders for editing values may need to change if more pacing functions are needed. |

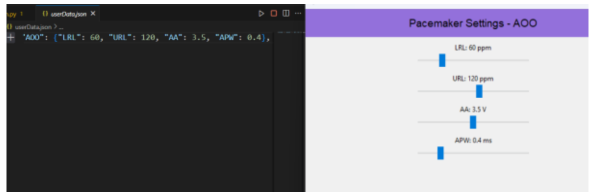
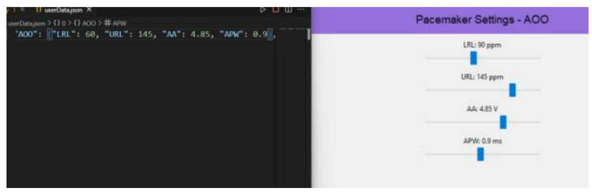
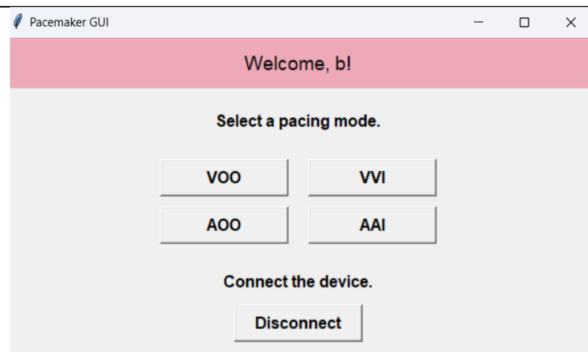
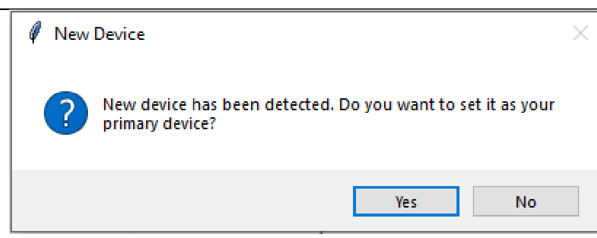
*Table 20: Design Decisions and Likely Changes*

## Testing

Every requirement was tested to validate the proper functioning of the DCM and validation of every requirement. Each test is documented below, detailing the nature of the test, the specific requirement it assesses, a visual representation of the test result, and a result of whether the test passed or failed.

| **Tests** | | | |
|---|---|---|---|
| **Test** | **Requirement it tests** | **Result** | **Pass/Fail** |
| Registering a new user:<br>username: test<br>password: user123 | DCM_3 |  | Pass |
| Trying to register a username that already exists. | DCM_3 |  | Pass |
| Pressing the login and register buttons on the home page redirects user to the correct page. | DCM_1<br>DCM_2<br>DCM_3 |  | Pass |

| | | | |
|---|---|---|---|
| Pressing the existing users button redirects the user to the existing users page, showing them all current users registered. | DCM_3<br>DCM_5 |  | Pass |
| In the existing users page, clicking on a button with the user's name delete's that user. | DCM_5 |  Before:  After:  | Pass |
| Trying to register more then 10 users prevents you from registering. | DCM_5 |  | Pass |
| Logging in with valid credentials redirects the user to the pacemaker interface. | DCM_3<br>DCM_6 |  | Pass |
| When no users are registered and there is no local file for storage, a new storage file is created with the first user registered. | DCM_4 | userData.json file was created with all of the parameters.  | Pass |
| Clicking on a pacing mode button redirects you to their respective page with all of it's correct programmable parameters ready for review/modification. | DCM_6<br>DCM_7 |  | Pass |

| | | | |
|---|---|---|---|
| | | Results of AAI mode. All other pacing modes passed. | |
| When programmable parameters are modified and saved in a pacing mode, the storage saves that modification. | DCM_7<br>DCM_4 | Before:<br><br>After:<br> | Pass |
| When the connect button is clicked with the same pacemaker device, the button switches to “disconnect”.<br>(No connection yet as only front-end required for A1) | DCM_8 |  | Pass |
| When the connect button is clicked with a new pacemaker device, the user is notified that the device is new, with options to connect or disconnect. | DCM_8 |  | Pass |

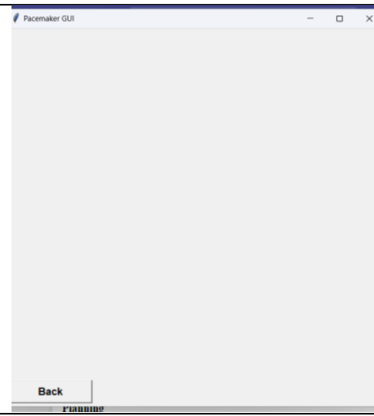
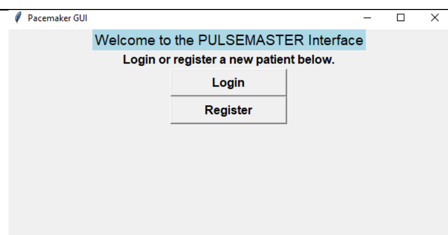
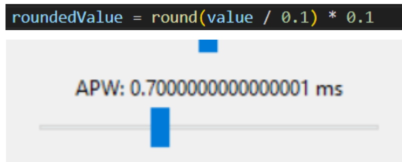
| | | | |
|---|---|---|---|
| When the egram button is clicked, the user is redirected to the egram page. (empty as only front-end is required for A1) | DCM_9 |  | Pass |

*Table 21: DCM Tests*

## Failures

During the development of the DCM, the team encountered a series of setbacks and failures on the path to refining the design. Document below is an account of each failure, describing the specific challenges faced, and outlining the measures taken to overcome the issue.

| **Failures** | | |
|---|---|---|
| **Failure** | **Visual of failure** | **Solution** |
| Components of each interface would not be centred, or have space between each component, making the UI/UX worse. |  | Added padding for each component that added height to each component. Added a parameter called "expand" that force the components to expand around the center. |
| Sliders current value would not update, when it was modified by the user. | No visualization since code would crash. error message: **Recursion Error: maximum recursion depth exceeded while calling a Python object** | The value of the slider was held as a string. We needed to convert it a float first in order to modify and save it.<br>`def VAAmpChange(value, VorA):`<br>`    value = float(value)`<br>`    if value < 0.5:`<br>`        roundedValue = 0` |
| Sliders had error showing the current value, adding a small remainder to the display | `roundedValue = round(value / 0.1) * 0.1`<br> | Rounding once rounded the value to an acceptable range, but still provided a remainder. We rounded twice, forcing it to have only 1 decimal digit.<br>`roundedValue = round(round(value / 0.1) * 0.1,1)` |

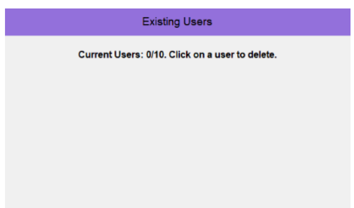
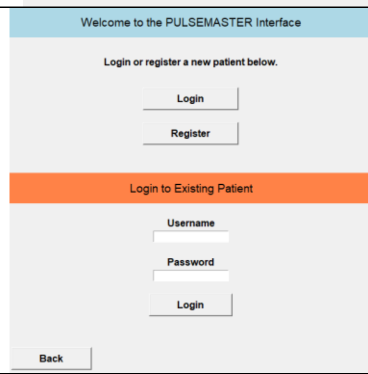
| | | |
|---|---|---|
| Sliders did not allow max value in acceptable range | When error occurs, code would crash and remove the sliders, producing an error message: **UnboundLocalError: cannot access local variable 'roundedValue' where it is not associated with a value** | Before:<br>`elif 90 <= value < 175:`<br>After:<br>`elif 90 <= value <= 175:`<br>The if statement did not include the max value, which was 175 for this specific slider. We added an = sign to account for it. |
| userData was the same across different user devices, when it should be a unique local storage for each device (laptop). | No visual representation. The root of the error was the userData.json being tracked in git, which made every user share the same storage. | The solution was to delete the .json file that was already pushed to our repository prior to the gitignore being created by accident. This allowed each user to store a unique untracked user data file. |
| In the existing users page, the command set up for each user button was run immediately when the button was loaded, deleting the user immediately. | All users are delete upon entering the page:<br> | To allow parameters in a function for the command variable, it MUST be set up as a lambda function:<br>`command=lambda name=user["username"]: deleteUser(name)` |
| When the user was redirected to a new page, the old page would not clear up. |  | Created a function called redirectPage() that would go through every current component in the GUI, and delete them. |

*Table 22: Failures table*

## Interface Visuals

Provided below are a series of visuals for each page of the DCM interface, in order to help understand the design.

Starting Page:

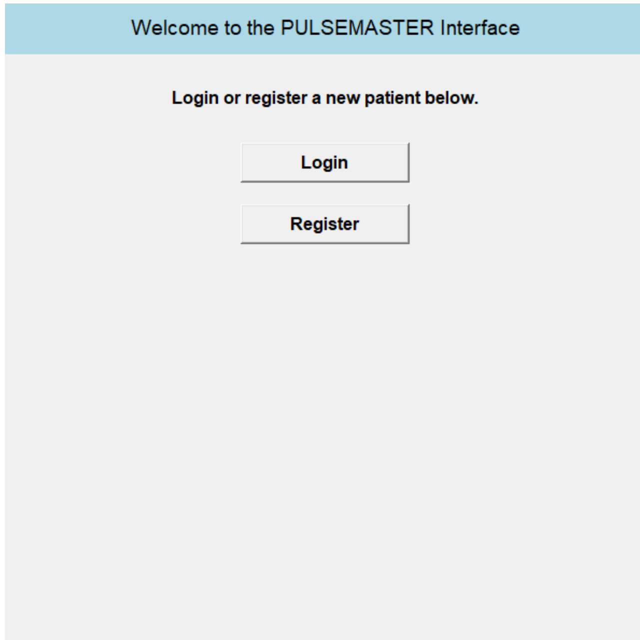



*Figure 15: Starting Page*

Login Page:

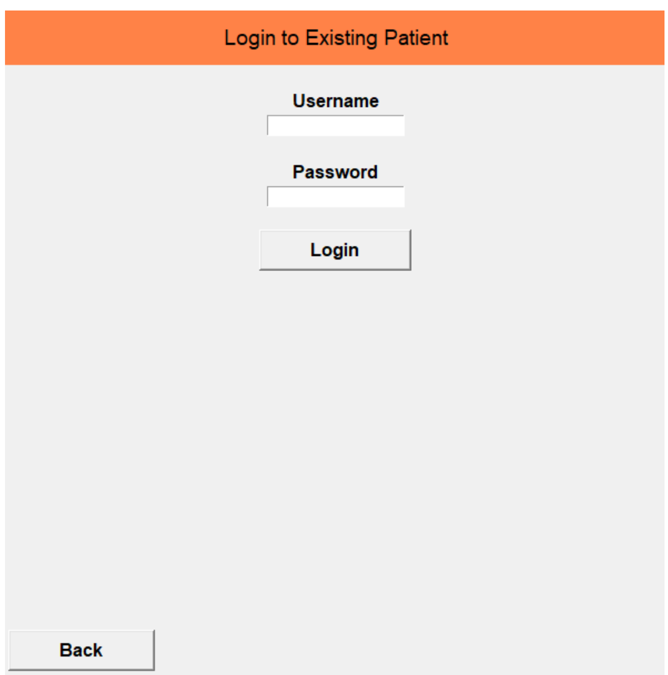



*Figure 16: Login Page*

Register Page:

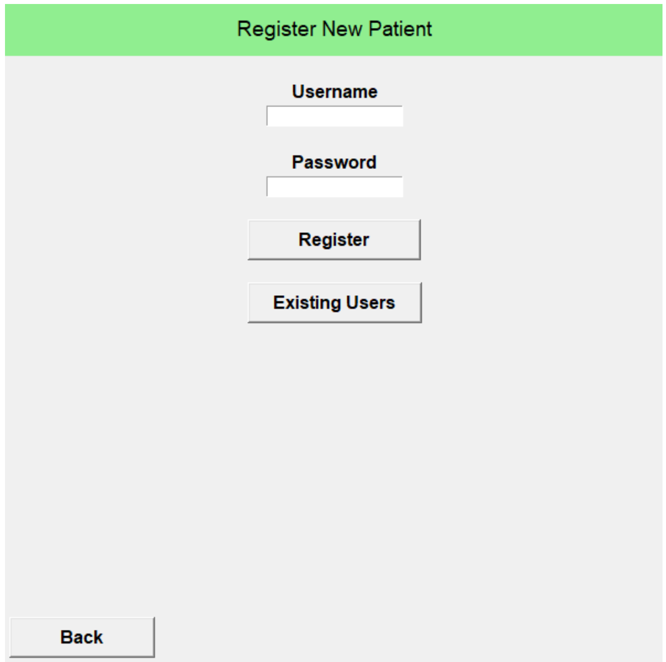



*Figure 17: Register Page*

Existing Users Page:

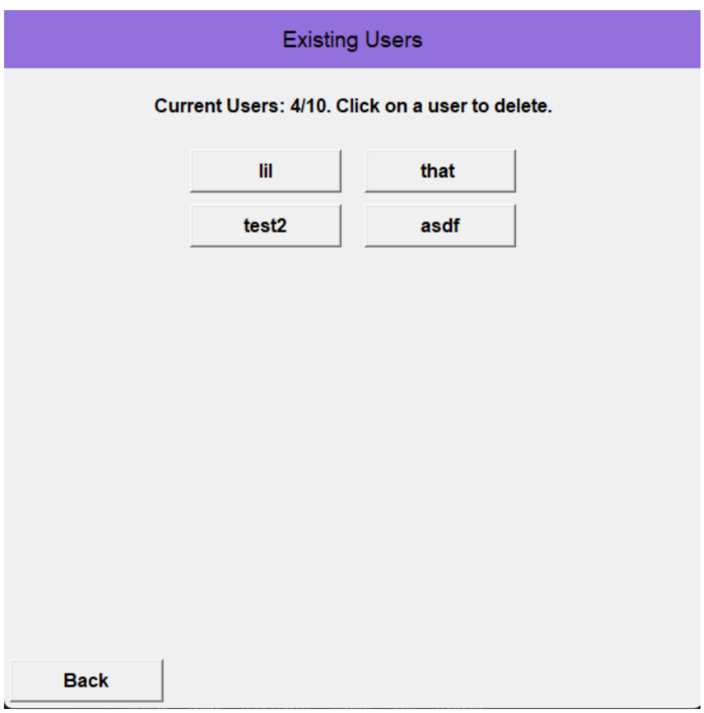



*Figure 18: Existing Users Page*

Home page:

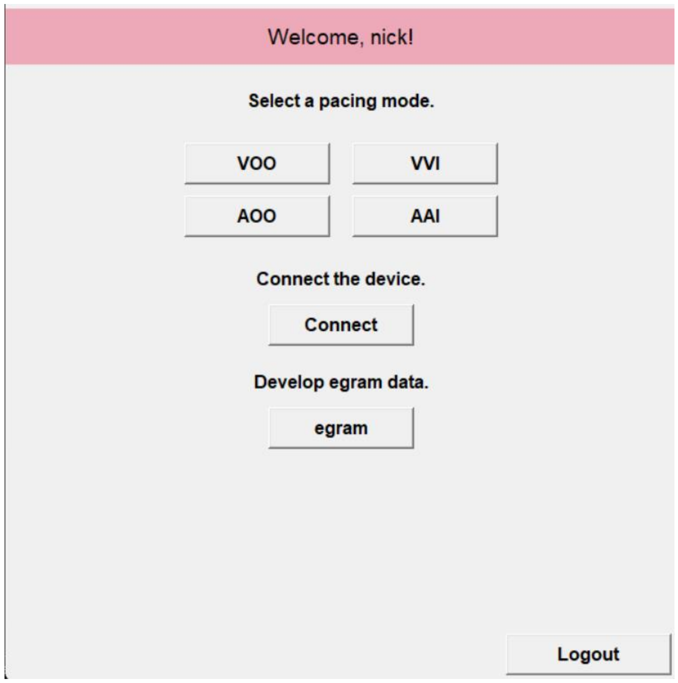



*Figure 19: Home Page*

VOO Page:

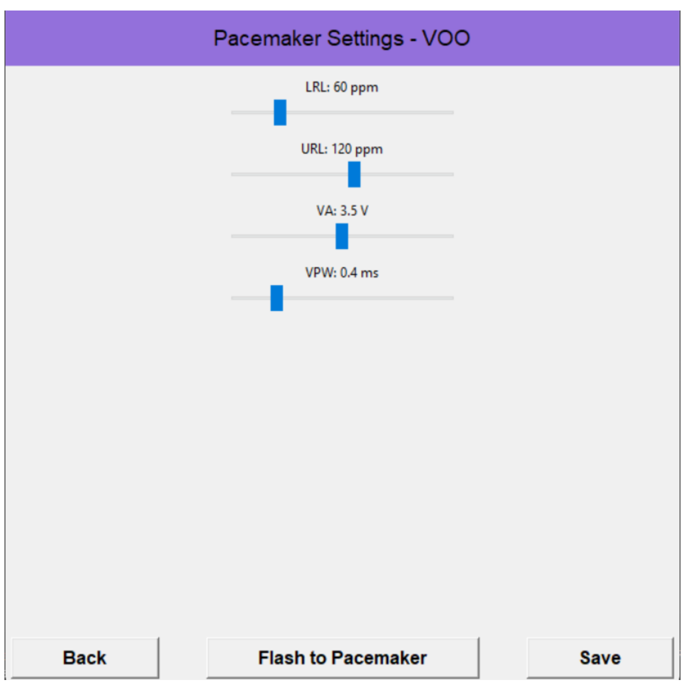



*Figure 20: VOO Page*

AOO Page:


*Figure 21: AOO Page*

VVI Page:

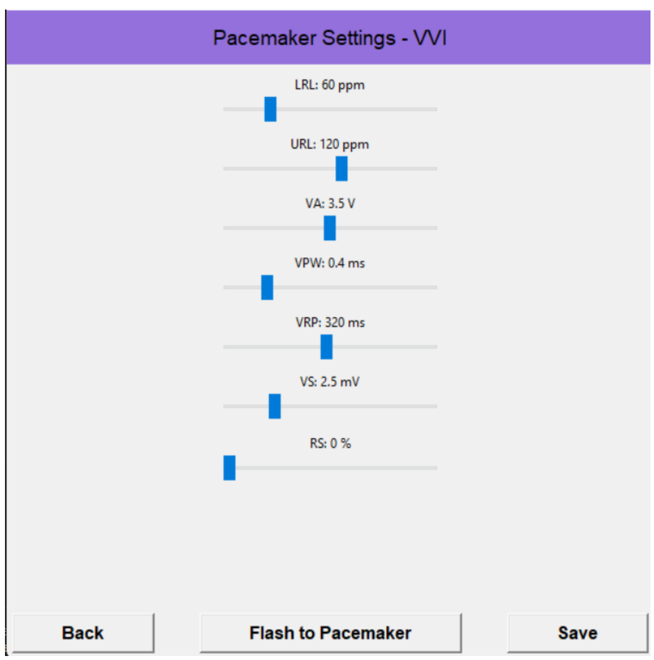



*Figure 22: VVI Page*

AAI Page:

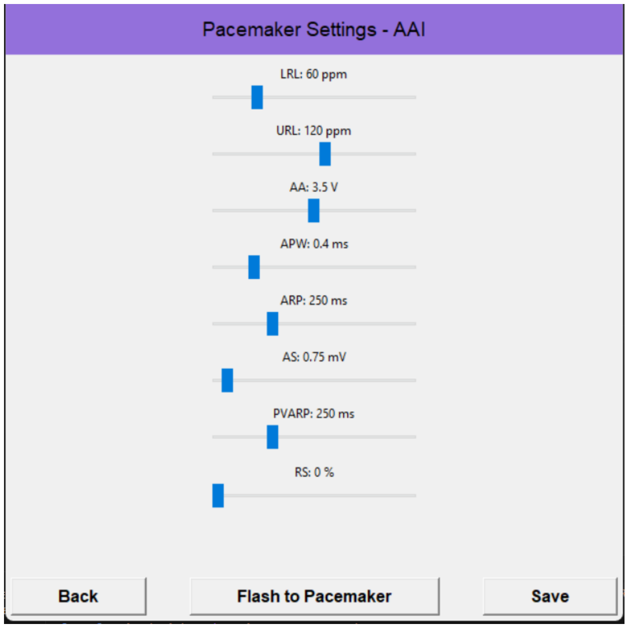



*Figure 23: AAI Page*