# Assignment 1 – Documentation

# MECHTRON 3K04

# Fall 2023


**Group 23 Members:**

Zihan Wang (wangz726)

Jane D'Souza (dsouzj22)

Yuxuan Xu (xu216)

Yujia Guo (guo71)

Fan Mo (mof5)

# Table of Contents

# Table of Tables



# Table of Figures

# Part 1 – Pacemaker Design

## 1.1 Requirements

### 1.1.1 Overall Pacemaker System

The pacemaker system is composed of three primary components: the Device (pulse generator), Device Controller-Monitor (DCM), and associated Leads [1]. The pacemaker should provide dual chamber, rate adaptive bradycardia pacing support and satisfy patients' need of implantation, ambulatory, follow-up and explanation. Furthermore, the radycardia analysis capabilities should encompass pacing-related measurements like lead impedance, pacing threshold, P and R wave measurement, battery status, temporary brady pacing, motion sensor trending, and tests to be performed. The DCM serves as the primary interface for communication with the pulse generator, providing features like diagnostics and sensor history. The leads, implanted in the patient, sense cardiac electrical signal, and deliver pacing therapy. Overall, the pacemaker system is a comprehensive solution for managing cardiac conditions through non-invasive control.

### 1.1.2 Device Operation

The device oversees and maintains a patient's heart rate. It also identifies and administers treatment for bradycardia conditions. Furthermore, it offers customizable pacing options in both single and dual chambers, for both permanent and temporary use. When operating in adaptive rate modes, an accelerometer is utilized to gauge physical activity, thereby indicating the pacing rate for the heart. The device is programmed and communicated with through two-way telemetry from the Device Controller-Monitor (DCM), enabling physicians to make non-invasive adjustments to the device's operating mode or parameters after implantation. The device should provide history data such as output rate histograms (atrial and ventricular) and sensor output data.

### 1.1.3 Pulse Pacing

The device should result in pulses with programmable voltages and widths for atrial and ventricular, which provide electrical heart-pacing stimulation. Both amplitude and width for pulse pacing should be independently programmable. Bipolar electrodes and a sensing circuit are expected to facilitate rate sensing. The determination of heart rate should rely on the measured cardiac cycle data from the detected rhythm and be evaluated on a per-interval basis.

### 1.1.4 Operation Modes

At present stage, only 4 modes are required to be selected by the DCM system. The 4 modes are AOO, VOO, AAI and VVI.

#### *1.1.4.1 AOO*

In AOO mode the pacemaker must include a lower rate limit, an upper rate limit, atrial amplitude control, and atrial pulse width control.

#### *1.1.4.2 VOO*

In VOO mode the pacemaker must include a lower rate limit, an upper rate limit, ventricular amplitude, and ventricular pulse width.

#### *1.1.4.3 AAI*

In AAI mode the pacemaker must include a lower rate limit, an upper rate limit, atrial amplitude control, atrial pulse width control, atrial sensitivity, ARP, and PVARP.

#### *1.1.4.4 VVI*

In VVI mode the pacemaker must include a lower rate limit, an upper rate limit, ventricular amplitude, ventricular pulse width, ventricular sensitivity, and VRP.

### 1.1.5 Programmable Parameters

| Parameter | Programmable Values | Increment | Nominal | Tolerance |
|---|---|---|---|---|
| Modes | AOO | – | DDD | – |
| | VOO | | | |
| | AAI | | | |
| | VVI | | | |
| Lower Rate Limit | 30-50 ppm | 5 ppm | 60 ppm | ± 8 ms |
| | 50-90 ppm | 1 ppm | | |
| | 90-175 ppm | 5 ppm | | |
| Upper Rate Limit | 50-175 ppm | 5 ppm | 120 ppm | ± 8 ms |
| A or V Pulse Amplitude Regulated | Off, 0.5 – 3.2 V | 0.1 V | 3.5V | 12 % |
| | 3.5 – 7.0 V | 0.5 V | | |
| A or V pulse Width | 0.05 ms | – | 0.4 ms | 0.2 ms |
| | 0.1 – 1.9 ms | 0.1 ms | | |
| Ventricular Refractory Period | 150 – 500 ms | 10 ms | 320 ms | ± 8 ms |
| Atrial Refractory Period | 150 – 500 ms | 10 ms | 250 ms | ± 8 ms |

*Table 1: Simulink Programable Parameters*

## 1.2 Design Decisions

A subsystem block is used to map all the hardware output pins for use in the program to their names as defined in Table 1 of Pacemaker Shield Explained. Another subsystem is used to map the hardware input pins to their names in the same table. We also created a subsystem block which contains all the Serial Input ports for future communication with the DCM system, although this subsystem block is not used for assignment 1. This allows the variables we used within the program to be more readable. 4 state charts were used for the 4 operation modes that are required and a multiport switch is used for mode selection. Gains of 20 were used to convert the input A or V amplitude from the voltage number between 0 to 5 volts to a PWM duty cycle number between 0 to 100. Annotation is added to all the blocks using the bolded red font to make the whole Simulink model more readable.

## 1.3 Simulink Diagram

The Simulink diagram must include necessary annotation to understand the model. There should be screenshots of the Simulink model in the documentation.

**AOO mode:**

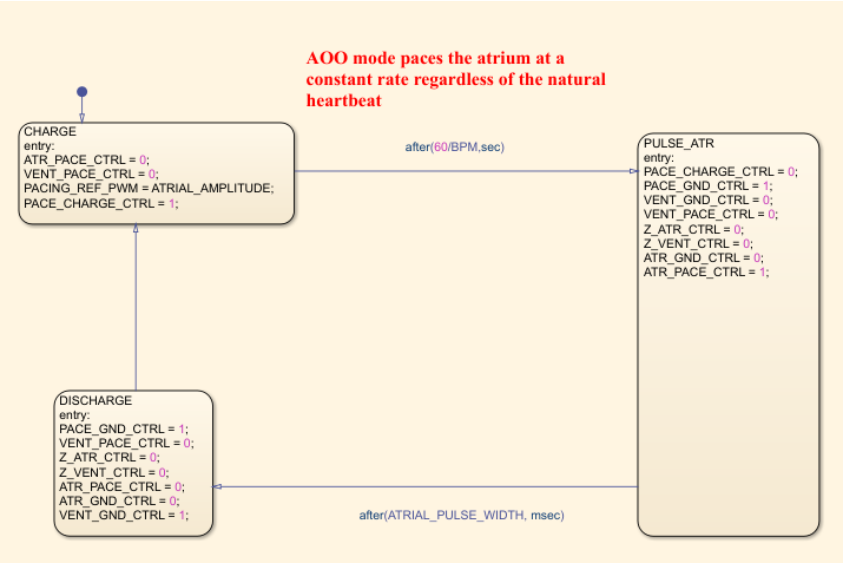



*Figure 1: AOO mode Stateflow*

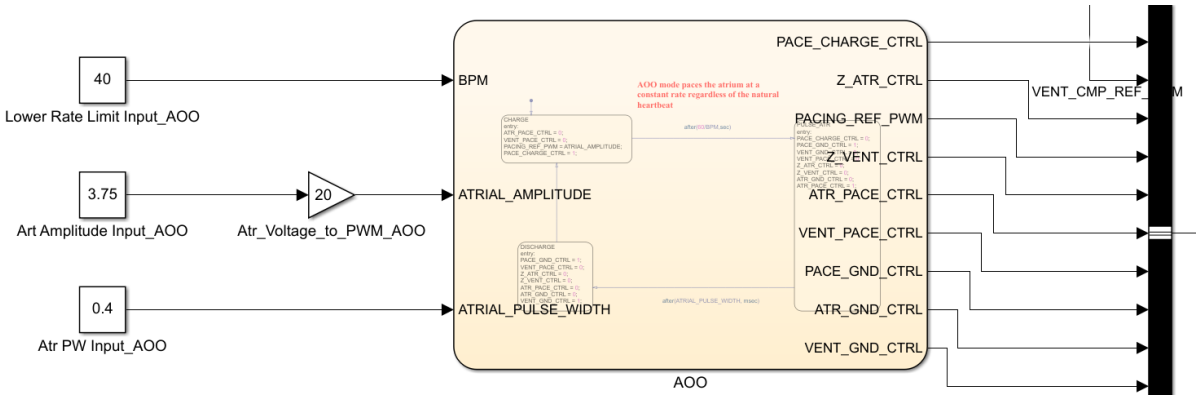



*Figure 2: Inputs of AOO mode*

These parameters (BPM, ATRIAL_AMPLITUDE, and ATRIAL_PULSE_WIDTH) are programmed with specific numerical values and are used in the charging and discharging processes of the pacemaker, which delivers the appropriate electrical signals to the hardware outputs. To charge C22, PACE_CHARGE_CTRL is set to 1 to prepare the pacing operation. Once the pacing capacitor is charged (60/BPM, sec), the system requires pacing in the atrium by setting PACE_GND_CTRL and ATR_PACE_CTRL to 1. The capacitor C21 is discharged by grounding the paced chamber to ensure a net zero current and protect the patient after pacing occurs.

**VOO mode:**

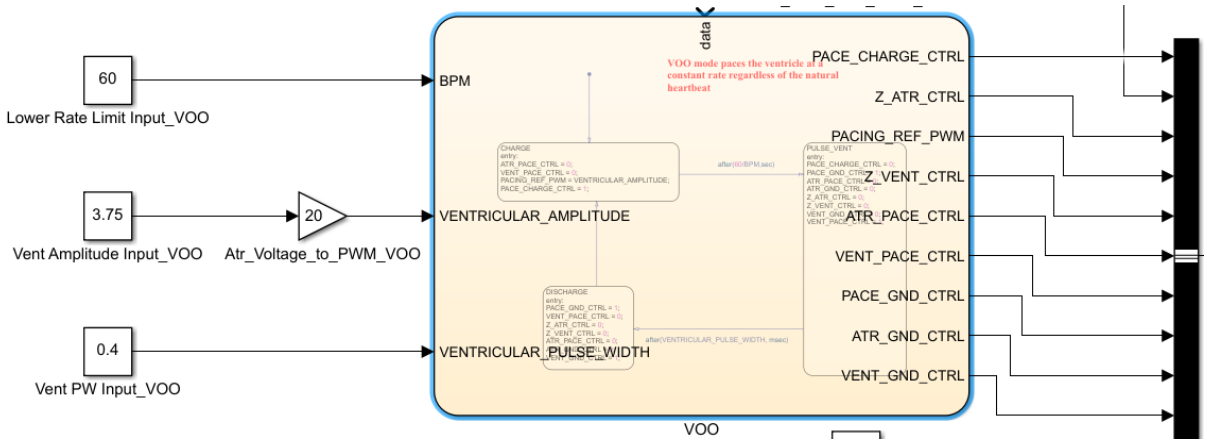



*Figure 3: Inputs of VOO mode*

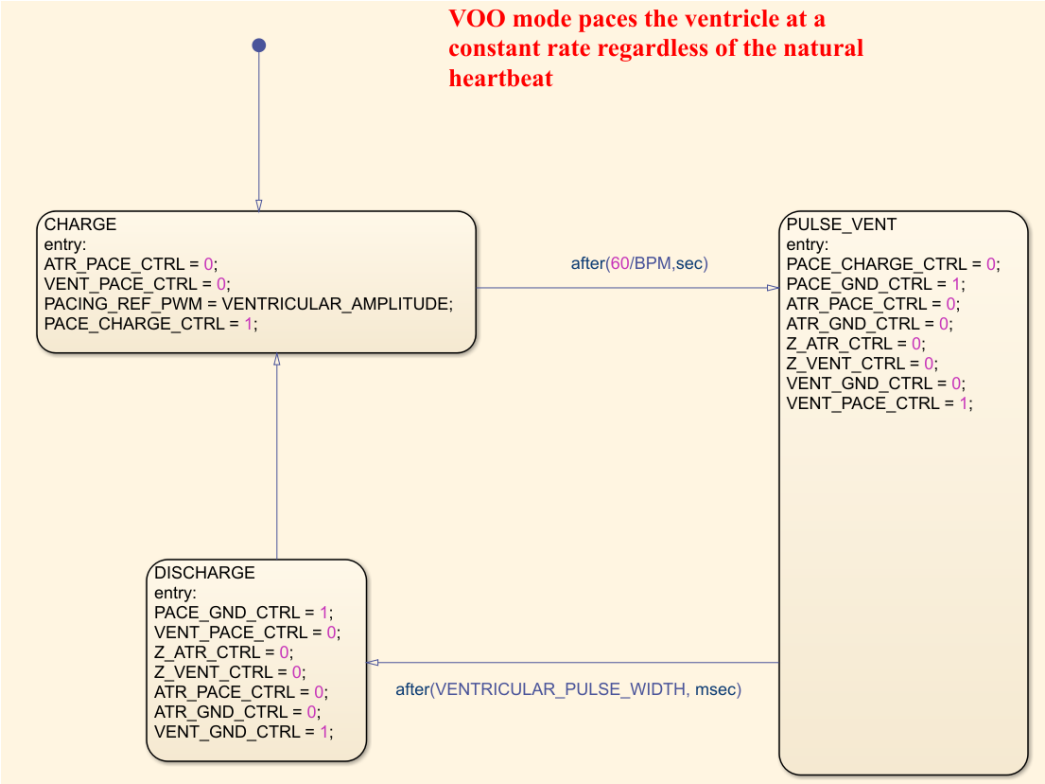



*Figure 4: VOO mode Stateflow*

The process of VOO mode is similar to the process of AOO mode except for locations of chamber. These parameters and inputs of VOO mode are the same as the AOO modes. In VOO mode, both the pacing signal and the sensing signal occur in the heart's ventricles, which emits a consistent signal at fixed intervals to stimulate the ventricles and maintain rhythm. Additionally, if the ventricles can generate their own heartbeat signals, the pacemaker stops from delivering pacing pulses.

**AAI mode:**

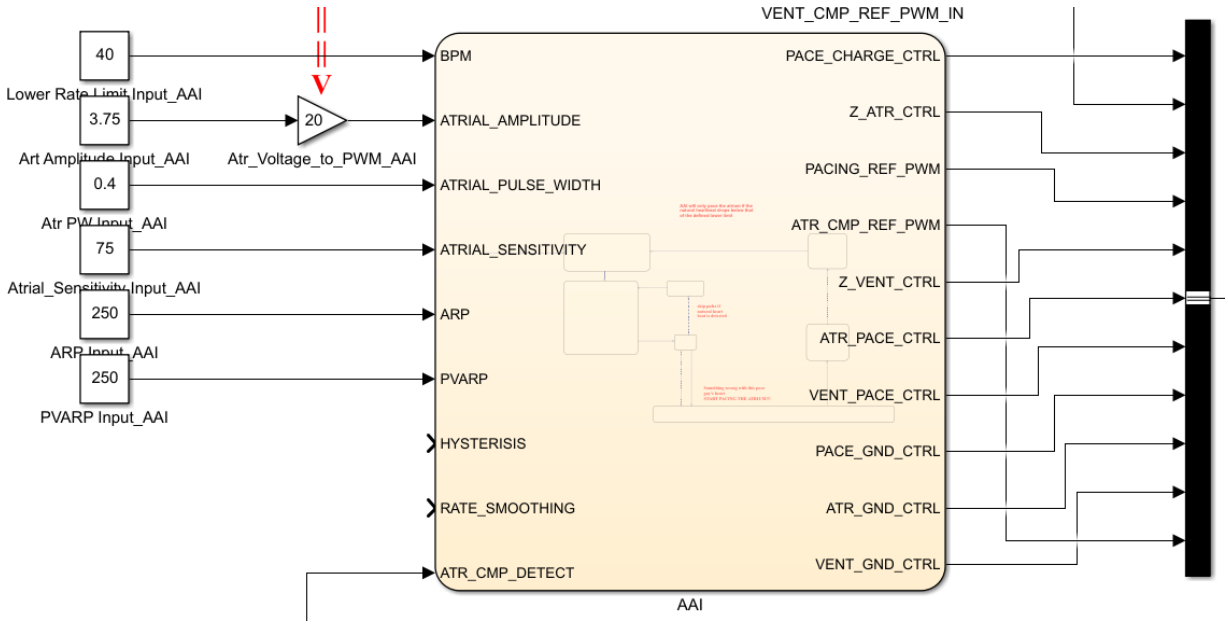



*Figure 5: Inputs of AAI mode*

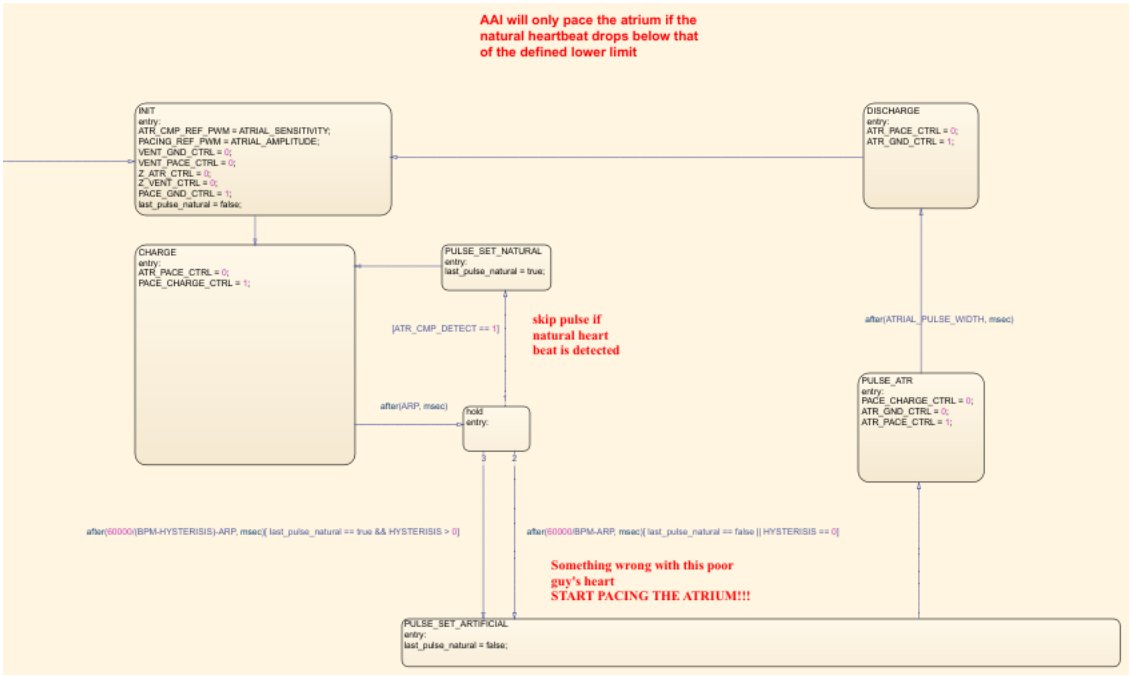



*Figure 6: AAI mode Stateflow*

At the initial state, the system set parameters of ATRIAL_SENSITIVITY and ATRIAL_AMPITUDE and preset the heartbeat is not produced naturally by the atrium (last_pulse_natural = false). The pacemaker system starts charging process (PACE_CHARGE_CTRL =1) to prepare for the next atrial pacing. After we determine the last heartbeat that was generated by the pacemaker or patients' atrium, the system initiates atrial pacing if last_pulse_natural = false and discharges to release capacitive charge (ATR_GND_CTRL =1).

**VVI mode:**

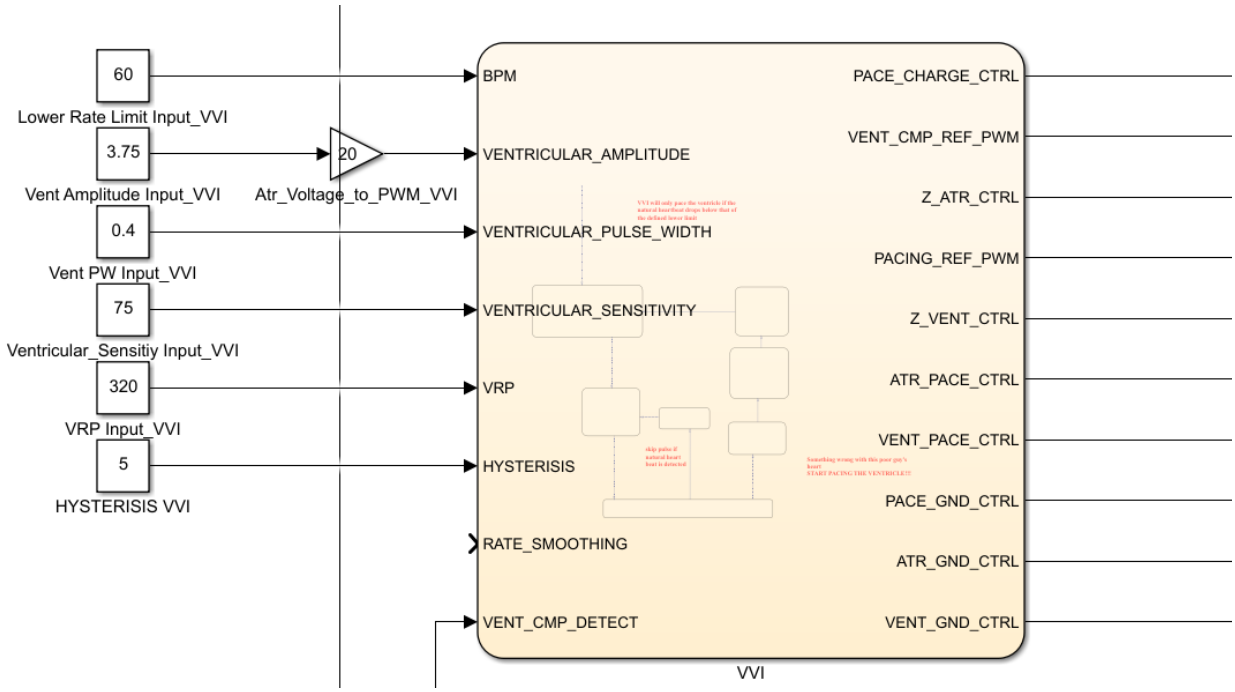



*Figure 7: Inputs of VVI mode*

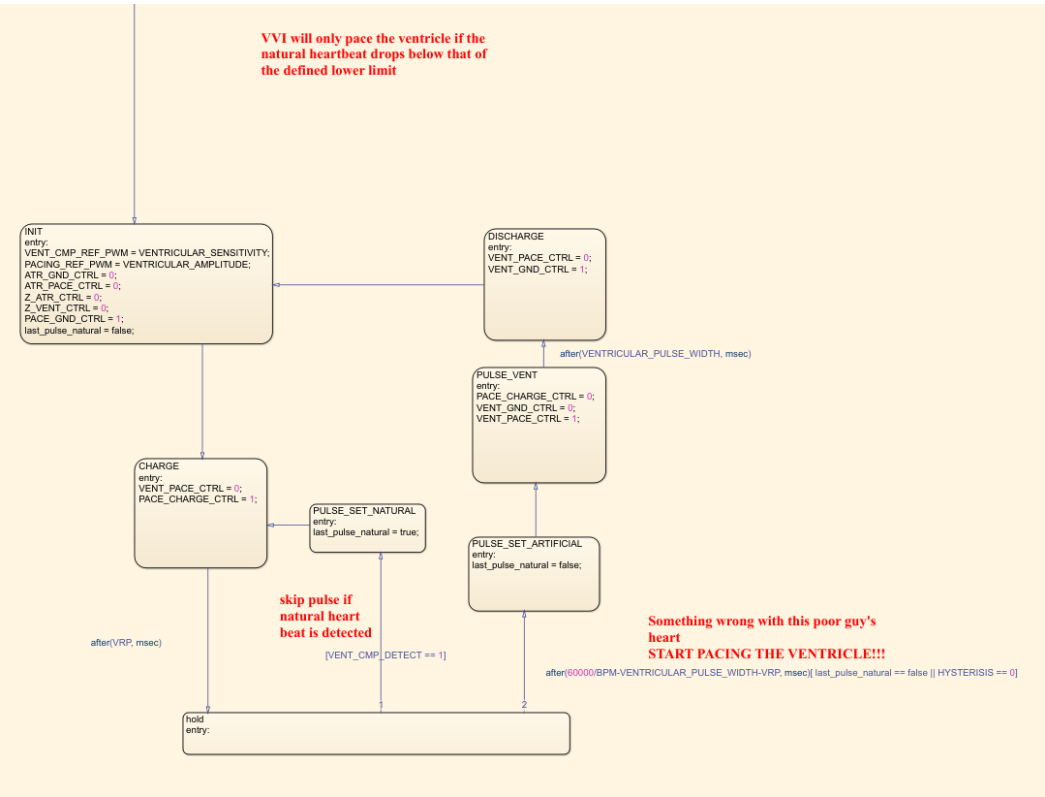



*Figure 8: VVI mode Stateflow*

The process of VVI mode is similar to the process of AAI mode. The main difference between these two modes is that the pacemaker primarily paces the ventricles to maintain the rhythm in VVI mode.

**Change the mode:**

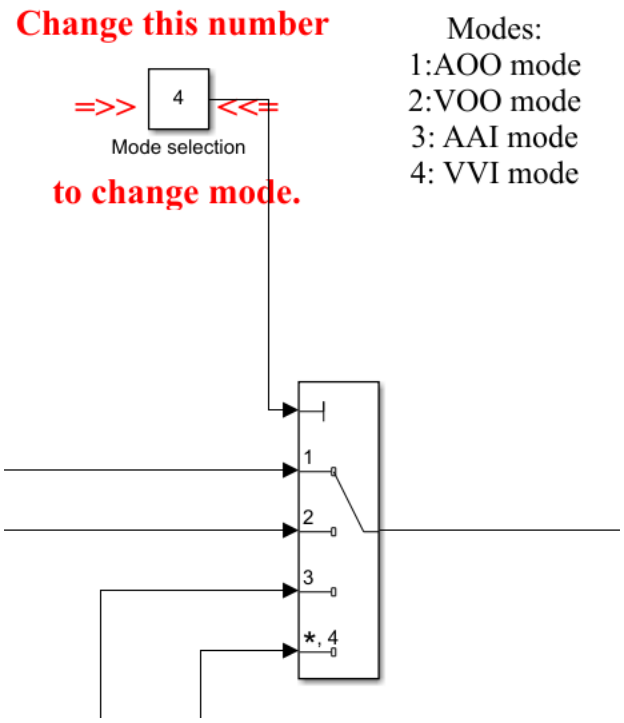



*Figure 9: Mode Selection Button*

The mode selection button is set to switch between pacemaker modes (AOO, VOO, AAI and VVI modes) ranging from 1 to 4.

**Hardware inputs:**

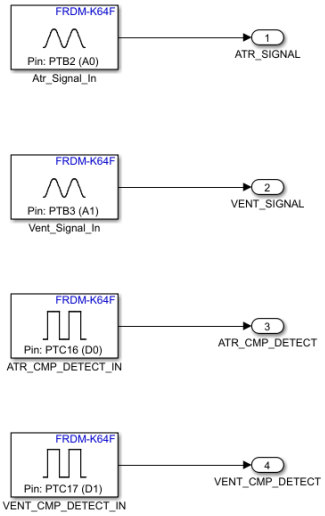



*Figure 10: Hardware Inputs*

These parameters read digital signals from the hardware and deliver them to the different modes as inputs. Each parameter is assigned pin based on the pin map of the K64F microcontroller.

**Hardware outputs:**

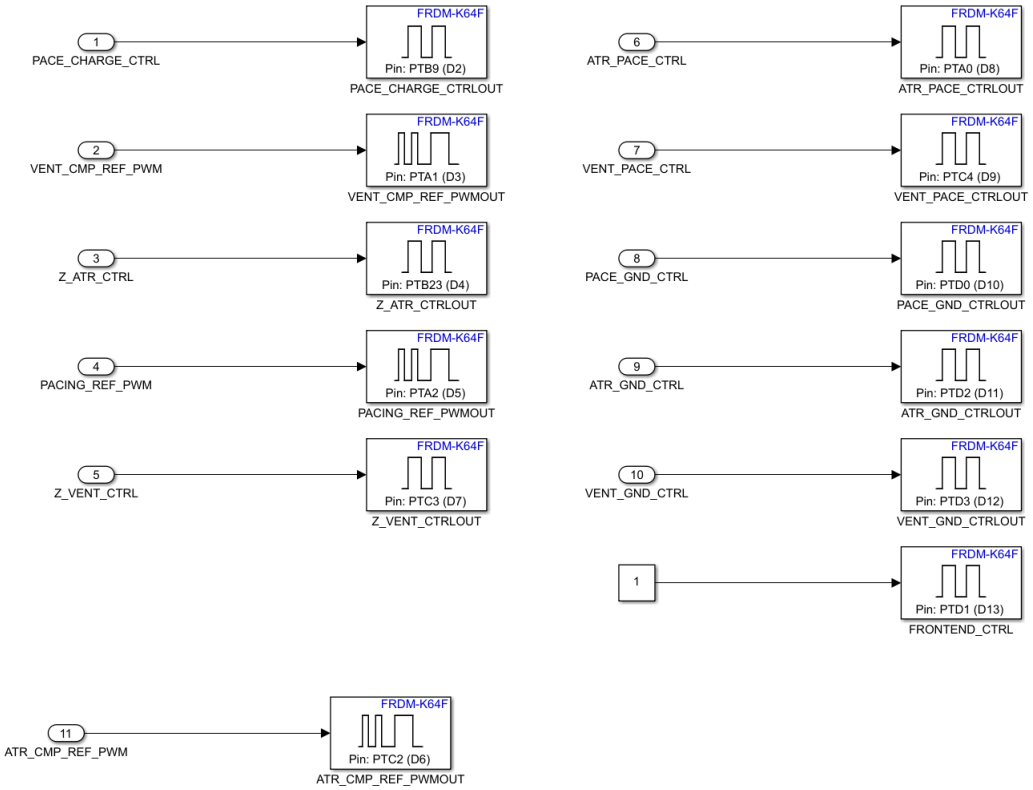



*Figure 11: Hardware Outputs*

The output data corresponds to the pins on the hardware, similar to the handling of input data. This separation allows for a clear situation between signal processing in the modes and the hardware.

## 1.4 Testing and Results

### *AOO Test Cases:*

**Report ID: AOO1**

Active Test Routine

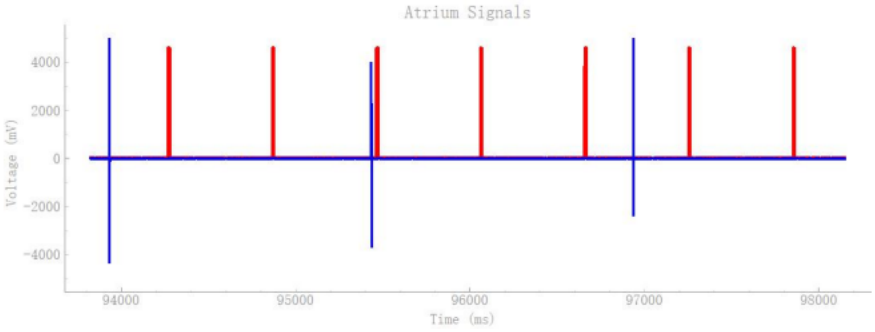
Atrium PW: 11.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 100 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

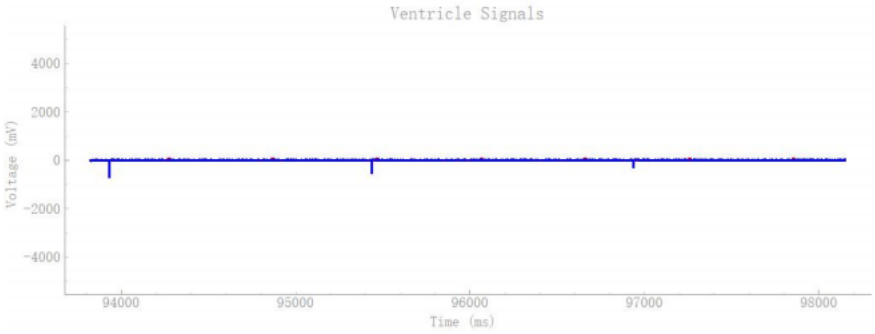



*Figure 12: AOO Test Case 1*

Condition: Patient with natural heartbeat

Expected Output: Pacemaker paces with atrium

Actual Output: Pacemaker paces as expected result

Pass/Fail: Pass

**Report ID: AOO2**

Active Test Routine

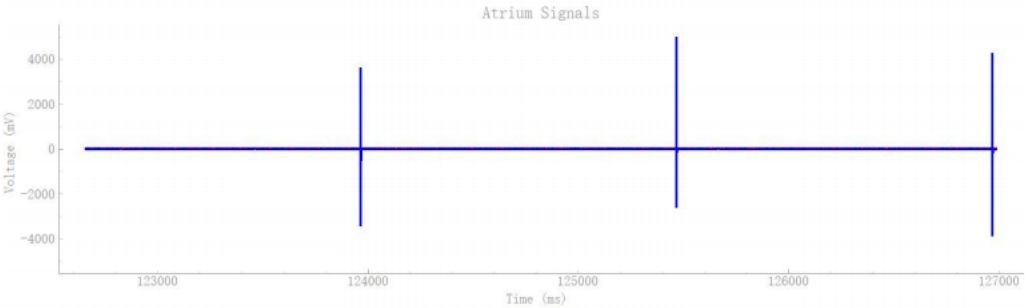
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

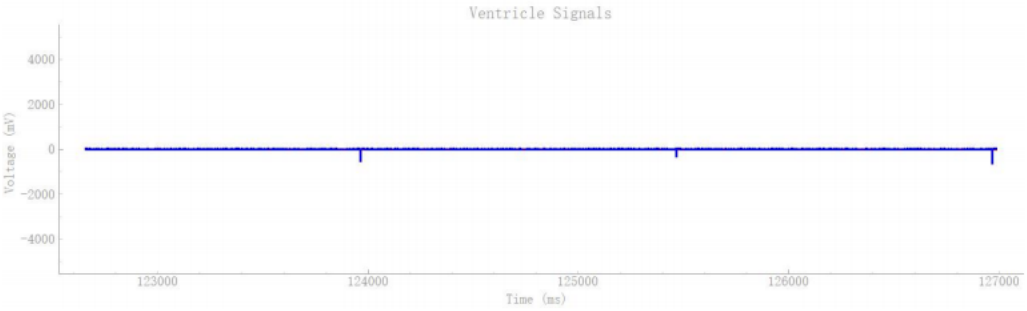



*Figure 13: AOO Test Case 2*

Condition: Patient without natural heartbeat

Expected Output: Pacemaker paces with atrium

Actual Output: Pacemaker paces as expected result

Pass/Fail: Pass

*VOO Test Cases:*

**Report ID: VOO1**

Active Test Routine

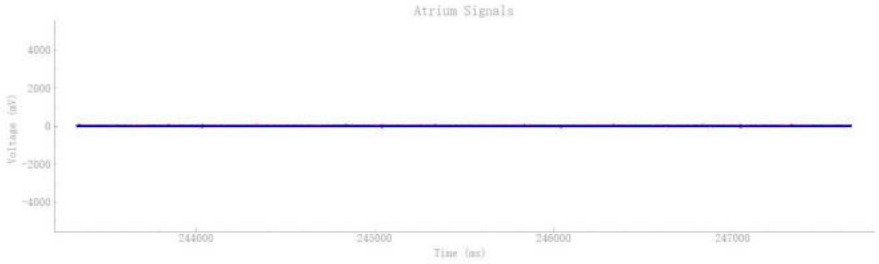
Atrium PW: 0.0 ms

Ventricular PW: 11.0 ms

Heart Rate: 120 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

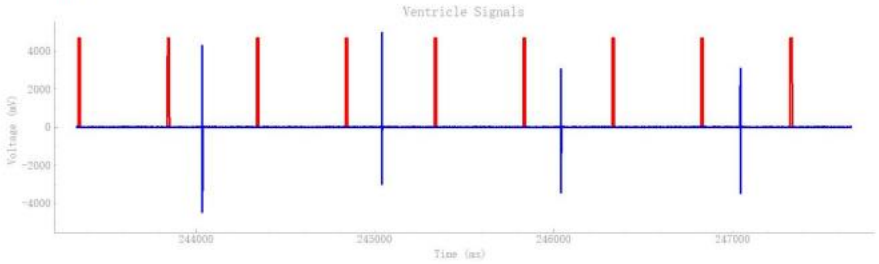



*Figure 14: VOO Test Case 1*

Condition: Patient with natural heartbeat

Expected Output: Pacemaker paces with ventricle

Actual Output: Pacemaker paces as expected result

Pass/Fail: Pass

**Report ID: VOO2**

Active Test Routine

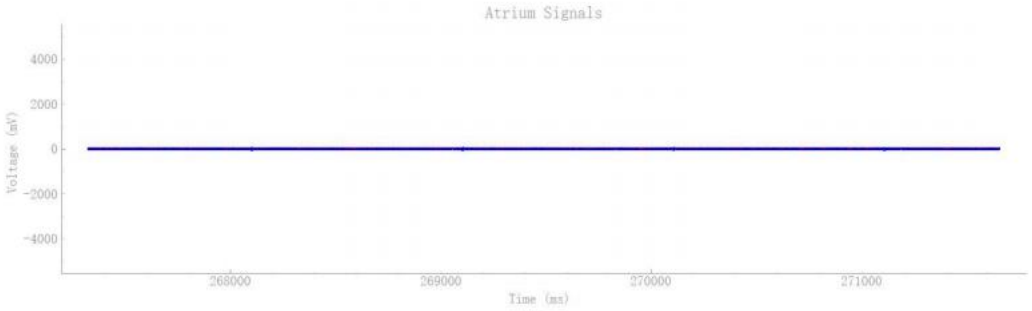
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 120 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

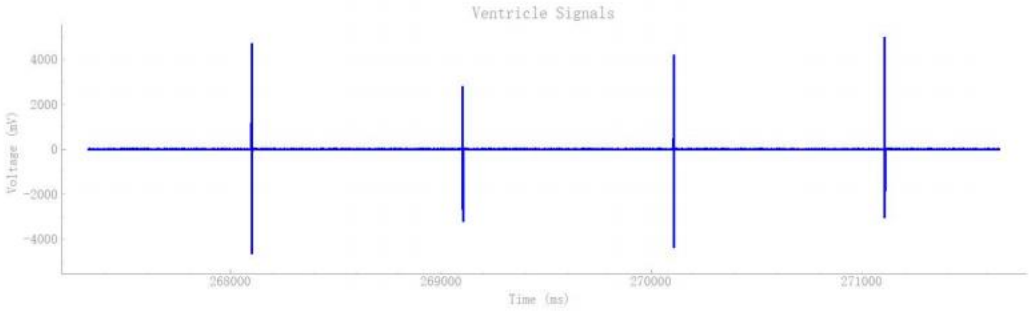



*Figure 15: VOO Test Case 2*

Condition: Patient without natural heartbeat

Expected Output: Pacemaker paces with ventricle

Actual Output: Pacemaker paces as expected

Pass/Fail: Pass

*AAI Test Cases:*

**Report ID: AAI1**

Active Test Routine

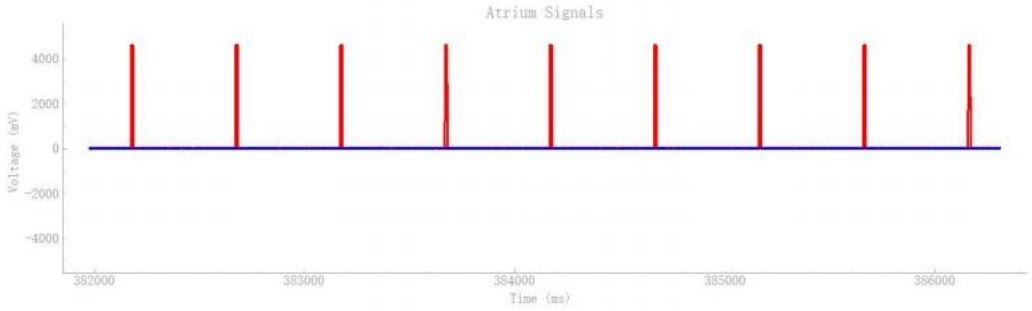
Atrium PW: 11.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 120 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

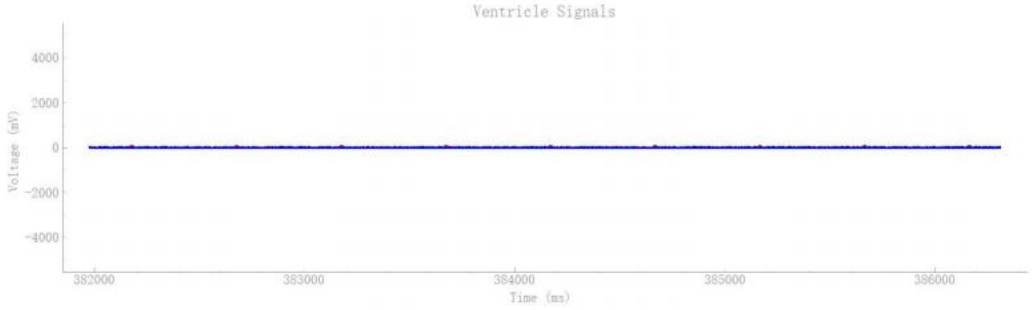



*Figure 16: AAI Test Case 1*

Condition: Patient with natural heartbeat

Expected Output: Pacemaker do not pace with atrium

Actual Output: Pacemaker do not pace with atrium

Pass/Fail: Pass

**Report ID: AAI2**

Active Test Routine

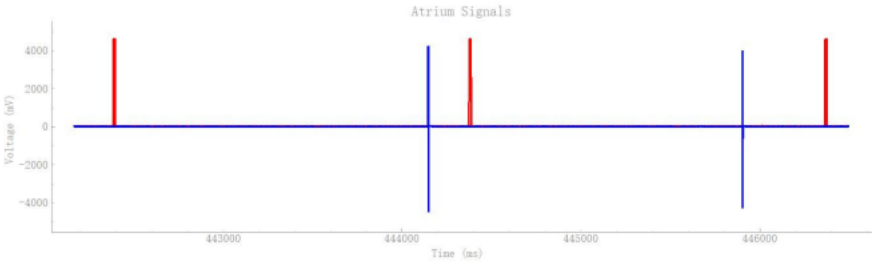
Atrium PW: 11.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

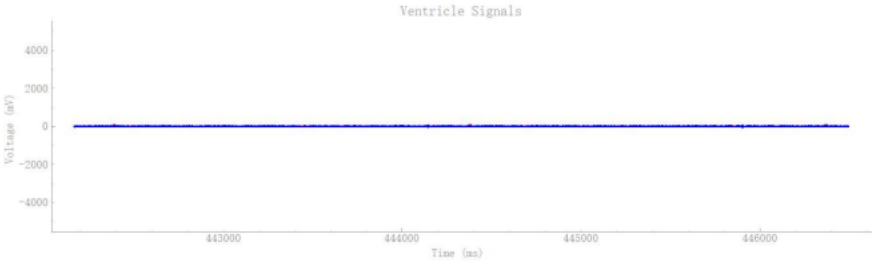



*Figure 17: AAI Test Case 2*

Condition: Patient with low frequency heartbeat

Expected Output: Pacemaker paces with atrium

Actual Output: Pacemaker paces as expected result

Pass/Fail: Pass

**Report ID: AAI3**

Active Test Routine

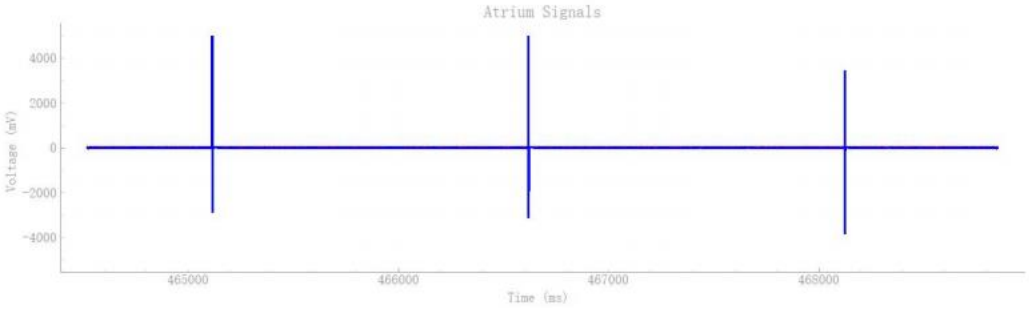
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

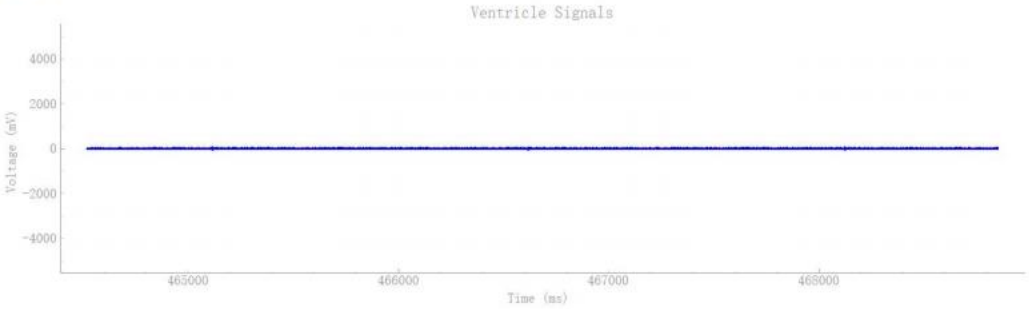



*Figure 18: AAI Test Case 3*

Condition: Patient without natural heartbeat

Expected Output: Pacemaker paces with atrium

Actual Output: Pacemaker paces as expected result

Pass/Fail: Pass

*VVI Test Cases:*

**Report ID: VVI1**

Active Test Routine

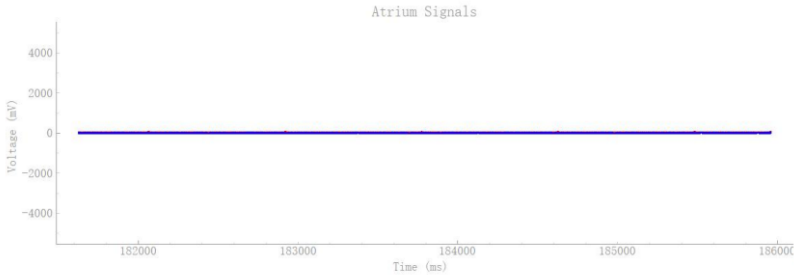
Atrium PW: 0.0 ms

Ventricular PW: 11.0 ms

Heart Rate: 70 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

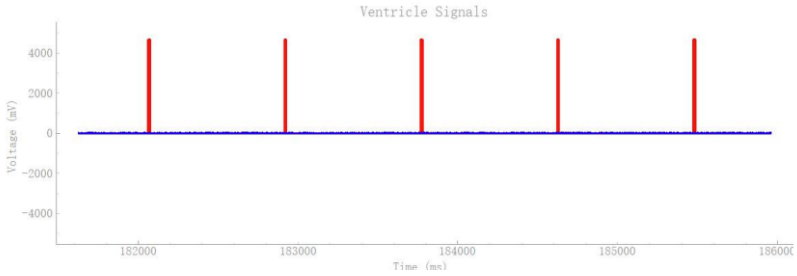



*Figure 19: VVI Test Case 1*

Condition: Patient with natural heartbeat

Expected Output: Pacemaker do not pace with ventricle.

Actual Output: Pacemaker do not pace with ventricle

Pass/Fail: Pass

**Report ID: VVI2**

Active Test Routine

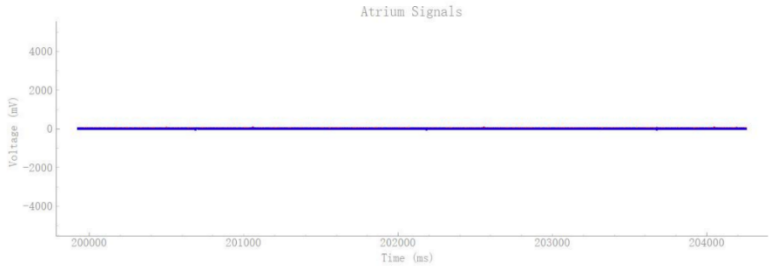
Atrium PW: 0.0 ms

Ventricular PW: 11.0 ms

Heart Rate: 40 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

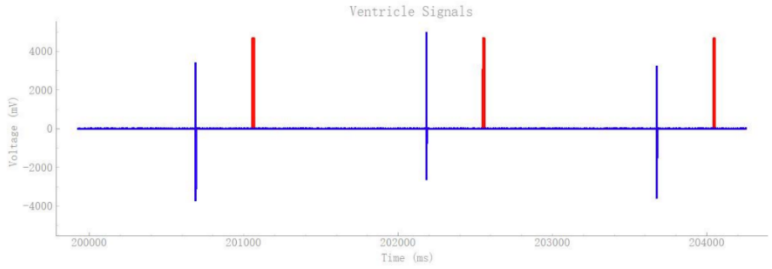



*Figure 20: VVI Test Case 2*

Condition: Patient with low frequency heartbeat

Expected Output: Pacemaker paces with ventricle.

Actual Output: Pacemaker paces as expected result

Pass/Fail: Pass

**Report ID: VVI3**

Active Test Routine

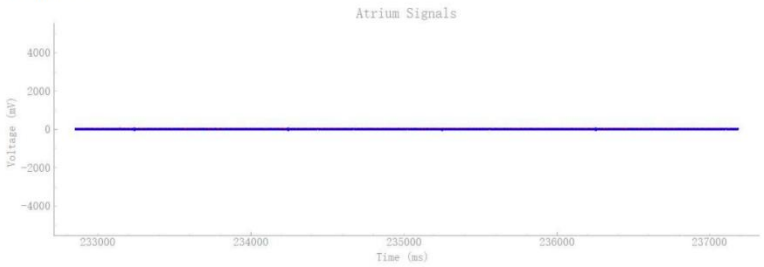
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:




Ventricle Plot:

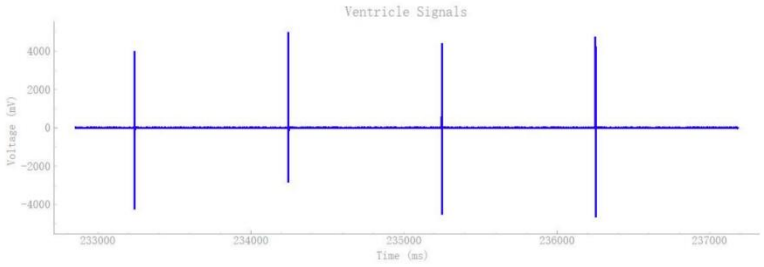



*Figure 21: VVI Test Case 3*

Condition: Patient without natural heartbeat

Expected Output: Pacemaker paces with ventricle.

Actual Output: Pacemaker paces as expected result

Pass/Fail: Pass

# Part 2 – DCM Design

Device Controller-Monitor (DCM):

A DCM is a device used by a medical professional to monitor and alter the behavior of the pacemaker device. It allows one to provide instructions (programming) and receive information (telemetry) from the pacemaker device [2].

## 2.1 Current Requirements

The requirement of this DCM includes a graphical user interface (GUI) that allows the physician (user) to get data and/or change data from the programmable pacemaker device. With the user prompt, the GUI is required to transmit a new set of instructions to alter the behavior of the embedded software in the pacemaker device by modifying programmable parameters or sending instructions to interrogate the pulse generator.

### 2.1.1 Welcome/login Screen

The interactive welcome screen interface allows the physician (user) to register or login as a user into the platform. Here the user can store data linked to their user account. A max of 10 users can be stored locally.

### 2.1.2 User Interface

Firstly, The DCM user interface should be capable of utilizing and managing windows for display of text and graphics [1]. Secondly, it is required that the user interface should be capable of processing user positioning and input buttons with the addition of displaying all programmable parameters for review and modification. It should also be capable of visually indicating when the DCM and the device are communicating and visually indicate when telemetry is lost due to the device being out of range (or lost) [1]. Lastly, the user interface should be capable of visually indicating when a different Pacemaker device is approached than was previously interrogated by the Physician (user) [1].

### 2.1.3 DCM Utility Functions

This part of the interface provides standard information and features for the user to conveniently use the interface. The required DCM utility functions is as follows:

DCM Utility Functions Chart

| Function Name | Features / Description |
|---|---|
| About | • Application Model Number<br>• Software revision number<br>• DCM Serial number<br>• Institution name |

| | |
|---|---|
| Set Clock | • Set date<br>• Set time |
| New Patient | • Interrogate new device without exiting the software application |
| Quit | • End a telemetry session |

*Table 2: DCM Utility Functions*

### 2.1.4 Printed Reports

The user will be able to print out reports to get information on data such as pacemaker parameters, test results, histograms and more. Each report is required to contain the following header information: Institution name, Date and time of report printing, device model and serial number, DCM serial number, Application model and version number, report name.

The following printed reports are available for the physician (user):

Printed Reports:

| Printed Report Name | Description |
|---|---|
| *Parameter / Status reports:* | |
| Bradycardia Parameters Report | - |
| Temporary Parameters Report | - |
| Implant Data Report | Storing lead implant date and polarity information. Stores pacing thresholds and P and R-wave amplitude functions |
| Threshold Test Report | Automatic pacing threshold test in AAI, VVI, and DDD modes of the DCM for both pulse width and amplitude measurements. |
| Measured Data Report | - |
| Marker Legend Report | - |
| Session Net Change Report | If parameter values are changed during the follow-up visit, the new setting is verified by viewing the "Session Net Change" report |

| | |
|---|---|
| Final Reports | This report will consist of the Measured Data, Threshold Test, Trending, Histograms, Implant Data, and Net Change reports. |
| *Bradycardia Diagnostic reports:* | |
| Rate Histogram Report | Involves the pacing rate and intrinsic rate distributions from a histogram recording period. Distributions shall be recorded for: paced atrial events, sensed atrial events, paced ventricular events, sensed ventricular events. |
| Trending Report | Involves pacing rate and sensor data |
| Electrogram | Available from the atrial and ventricular sense/pace leads and a surface electrogram. |

*Table 3: User Available Printed Reports*

### 2.1.5 Pacing Modes

The Bradycardia Operating modes are required in the DCM to be programmed. The operating/pacing modes implemented are: AAI, AOO, VOO, VVI.

### 2.1.6 Programmable and Measured Parameters

The user interface is required to display these programmable parameters. They are provided for controlling the delivery of patient-tailored bradycardia therapy. This chart includes the programmable parameters based on their nominal value, max and min programmable values, their increment and lastly their corresponding units.

Programmable and Measured Parameters in the interface

| Parameter | Nominal Value | Min Value | Max Value | Increment | Units |
|---|---|---|---|---|---|
| Programmable Parameters | | | | | |
| Lower Rate Limit | 60 | 30 | 175 | 5 | ppm |
| Upper Rate limit | 120 | 50 | 175 | 5 | Ppm |
| Maximum Sensor Rate | 120 | 50 | 175 | 5 | Ppm |

| | | | | | |
|---|---|---|---|---|---|
| Fixed AV Delay | 150 | 70 | 300 | 10 | ms |
| V Pulse Amplitude | 5.0 | 0 | 5 | 0.1 | V |
| A Pulse Amplitude | 3.5 | 0 | 5 | 0.1 | V |
| V Pulse Width | 1.0 | 0.1 | 1.9 | 0.1 | ms |
| A Pulse Width | 1.0 | 0.1 | 1.9 | 0.1 | ms |
| V Sensitivity | 1.0 | 1.0 | 10 | 0.5 | mV |
| Ventricular Refractory Period (VRP) | 320 | 150 | 500 | 10 | ms |
| Atrial Refractory Period (ARP) | 250 | 150 | 500 | 10 | ms |
| PVARP | 250 | 150 | 500 | 10 | ms |
| Activity Threshold | Med | - | - | V-Low, Low, Med-Low, Med, Med-High, High, V-High | - |
| Reaction Time | 30 | 10 | 50 | 10 | Sec |
| Response Factor | 8 | 1 | 16 | 1 | - |
| Recovery Time | 5 | 2 | 16 | 1 | min |
| Measured Parameters | | | | | |
| P and R wave measurement | - | - | - | 0.1 | mV |
| Lead Impedance | - | - | - | 50 | Ω |
| Battery Voltage | - | - | - | 0.01 | V |

*Table 4: Programmable and Measured Parameters*

### 2.1.7 Diagnostics and communication

The system will provide diagnostic tools such as: Measured data, Threshold Test, Rate training, Histogram and real-time data. P and R wave measurements will also be displayed when commanded by the DCM. The user also has the option of viewing real-time electrograms, either through the screen or from a printed copy. The user can access the following Internal electrogram (EGM) options: an atrial EGM, a ventricular EGM or both.

### 2.1.8 Potential changes to DCM requirements

There will be no potential changes for the DCM system. All programmable parameters/modes might be subject to change depending on future objectives, but this will be easy to implement due to the current structure of the DCM modules. This will allow us to easily integrate any new pacing modes or parameters that are required such as DOO (dual asynchronous pacing) or DDDR (dual rate adaptive demand pacing).

## 2.2 Design decisions

The choice to develop the Device Communication Module (DCM) GUI using Python and the tkinter library was made after careful analysis focused on the unique benefits and project-specific design specifications. Python was the best option for creating a user interface in a medical setting because of its readability and versatility. Cross-platform compatibility was critical because the DCM GUI needed to work flawlessly across Windows, macOS, and Linux, among other operating systems. A more simple and manageable design process was made possible by Python's clean syntax and simplicity, which is important in a medical application where accuracy and clarity are critical.

Python's broad library support was one of its main features, and tkinter, a standard GUI library, gave users access to a large collection of pre-made graphical elements including buttons and input fields. The lack of necessity to build these components from the ground up allowed for an acceleration of development thanks to this extensive toolset. Another strong argument in favour of Python's adoption was its lively and supportive community, which provides a plethora of information, documentation, and support from other members. In the highly regulated and standards-driven healthcare industry, these qualities proved to be invaluable in resolving issues and guaranteeing compliance with recommended procedures.

For the DCM GUI to communicate with external medical monitoring devices and backend systems, Python's smooth integration features were essential. Python's ability to seamlessly integrate with a wide range of technologies via libraries and APIs expedited communication across various medical system components. In the end, tkinter and Python were selected over alternative languages and libraries because they met the particular requirements of the project. Python is the safest option for a medical GUI because of its cross-platform compatibility, readability of code, speed of development, and strong community support—especially in a setting where accuracy, dependability, and standard compliance are crucial. This choice guaranteed the DCM GUI's effective and user-friendly design, matching it with the exacting standards of the healthcare industry.

### 2.2.1 Maximum Number of Users Allowed

The first design decision was to limit the number of registered users. As mentioned in the requirements, the number of users is limited to 10.

Code:

```
# Maximum number of users allowed
MAX_USERS = 10
```

### 2.2.2 User Authentication

The second design decision was to authenticate users based on their username and password. As seen in the figure below, the login function should check if the user is already registered or not. The register function should allow the user to create their unique username and password.

Code:

```
# Function to handle user login
def login():
# Function to handle user registration
def register():
```

### 2.2.2 User Data Storage

The third design decision was to store user data in JSON files for the login and register function described above to work properly.

Code:

```
# Function to load user data from a JSON file
def load_users():


# Function to save user data to a JSON file
def save_users(data):
```

### 2.2.3 User Variable Storage

The fourth design decision was to store user-specific variables in JSON files, one for each mode. The variables will be stored individually for each user.

Code:

```
# Functions to load and save user variables for a specific mode
def load_user_variables(username, mode):

def save_user_variables(username, mode, variables):
```

### 2.2.4 Mode Selection:

The fifth design decision was to allow users to select a mode (AAI, VVI, VOO, AOO).

Code:

```
# Function to show the mode selection page
def show_mode_select(username):
```

### 2.2.6 User Variable Display:

The sixth design decision was to display user-specific variables based on the selected mode. After the user has logged in, there should be a page where the user is able to select modes and their corresponding parameters.

Code:

```
# Function to show the welcome page with user-specific variables
def show_welcome_page(username, mode):
```

### 2.2.7 Variable Update:

The seventh design decision was to allow users to update their variables and save those updates after the function is executed.

Code:

```
def update_variables():
```

### 2.2.8 Main User Interface:

The eighth design decision was to create a graphical user interface (GUI) for user login and registration.

Code:

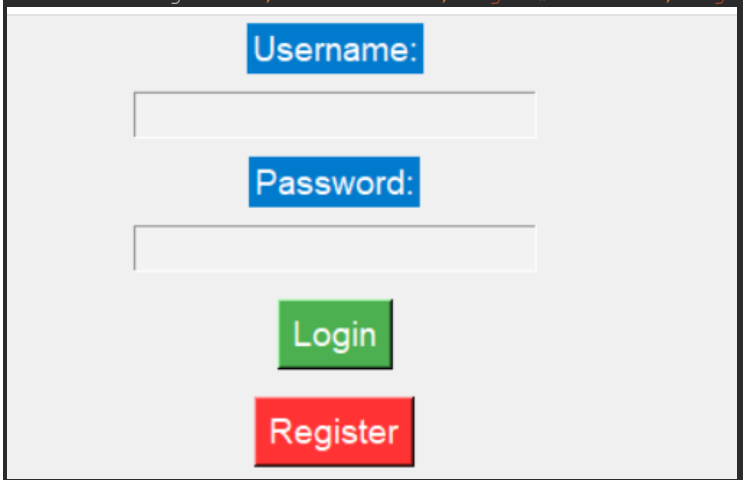
```
# Create the main login/registration window
login_window = tk.Tk()
```

### 2.2.9 GUI Elements:

The ninth design decision was to create labels, entry widgets, and buttons for user interaction.

Code:

```
username_label = tk.Label(login_window, text="Username:", font=font,
fg=label_color, bg="#007acc")

username_entry = tk.Entry(login_window, font=font, bg=entry_bg_color)

password_label = tk.Label(login_window, text="Password:", font=font,
fg=label_color, bg="#007acc")

password_entry = tk.Entry(login_window, show="*", font=font,
bg=entry_bg_color)

login_button = tk.Button(login_window, text="Login", command=login,
font=font, bg="#4CAF50", fg=label_color)

register_button = tk.Button(login_window, text="Register",
command=register, font=font, bg="#ff3333", fg=label_color)
```




### 2.2.10 DCM Connection Status Display:

The 10th design decision was to display DCM connection status.

Code:

```
dcm_label = tk.Label(login_window, text="DCM connected", font=font,
fg="green")
```

### 2.2.11 Pacemaker Detection Display:

The 11th design decision was to display the ID of the pacemaker currently being used.

Code:

```
PACEMAKER_label = tk.Label(login_window, text="PACEMAKER ID:1 is
currently being used", font=font, fg="green")
PACEMAKER_label.pack()
```

### 2.2.12 Egram data Display:

The 12th design decision was to display the Egram data tab of the pacemaker currently being used.

- For this assignment, this only works as a empty tab, for future assignments, this will be properly implemented.

Code:

```
# Function for the "Egrams" button (you can customize this function's
behavior)
def egrams_function():
    messagebox.showinfo("Egrams", "This is the Egrams feature. You can
customize its behavior here.")
```

## 2.3 Advanced design decisions:

The advanced design decisions for this medical system or pacemaker control application encompass a range of enhancements aimed at improving system functionality and user experience. First and foremost, a maximum limit of registered users, set at 10 in the code (MAX_USERS = 10), helps control system usage, preventing overuse. User data, such as usernames and passwords, is now stored in JSON files for persistent data storage. Functions have been implemented for loading and saving user data to and from these files. In addition, user-specific variables for each operational mode (AAI, VVI, VOO, AOO) are saved in separate JSON files. Functions enable the loading and saving of these variables, promoting a more personalized user experience. User authentication is another significant feature, requiring users to log in with their username and password. If credentials match stored data, access is granted; otherwise, a login failure message is displayed.

To facilitate user interaction and mode selection, a graphical user interface (GUI) has been introduced. This GUI includes labels, entry widgets, and buttons for user login and registration, enhancing the overall user experience. Furthermore, a DCM (Device Communication Module) connection status display informs users about the system's connection to external devices. When the DCM is connected, a green "DCM connected" label is presented to provide this critical feedback. These advanced design choices collectively make the medical system more robust, user-friendly, and equipped with the necessary tools for data management and pacemaker control while ensuring user security and personalization.

### 2.3.1 Maximum Number of Users Allowed

This decision was made to limit the number of registered users to prevent overuse of the system.

Code:

```
# Maximum number of users allowed
MAX_USERS = 10
```

### 2.3.2 User Data Storage

This advancement was made to store user data, including usernames and passwords, in JSON files for persistent data storage.

Code:

```
# Function to load user data from a JSON file
def load_users():
    try:
        with open("users.json", "r") as file:
            return json.load(file)
    except FileNotFoundError:
        return {}

# Function to save user data to a JSON file
def save_users(data):
    with open("users.json", "w") as file:
        json.dump(data, file)
```

### 2.3.3 User Variable Storage

This decision was made to store user-specific variables for each mode in separate JSON files.

Code:

```
# Function to load user variables from a JSON file for a specific mode
def load_user_variables(username, mode):
    try:
        with open(username + f"_{mode}_variables.json", "r") as file:
            return json.load(file)
    except FileNotFoundError:
        return {}

# Function to save user variables to a JSON file for a specific mode
def save_user_variables(username, mode, variables):
    # Load existing user variables for the selected mode
    existing_variables = load_user_variables(username, mode)

    # Update existing variables with new values
    existing_variables.update(variables)

    # Save user variables for the selected mode back to the JSON file
    with open(username + f"_{mode}_variables.json", "w") as file:
        json.dump(existing_variables, file)
```

### 2.3.4 User Authentication

This decision was made to authenticate users based on their username and password.

Code:

```
# Function to handle user login
def login():
    global current_user
    global dcm_connected  # Declare the global variable

    username = username_entry.get()
    password = password_entry.get()

    if username in users and users[username] == password:
        current_user = username  # Set the current user
        dcm_connected = True  # Set DCM connection status to True
        show_mode_select(username)
    else:
        messagebox.showerror("Login Failed", "Invalid username or
password")
```

### 2.3.5 Mode Selection

This decision was made allow users to select one of four modes (AAI, VVI, VOO, AOO).

Code:

```
# Function to show the mode selection page
def show_mode_select(username):
    # Destroy the current window
    if 'welcome_window' in globals():
        welcome_window.destroy()

    # Create the mode selection window
    mode_window = tk.Tk()
    mode_window.title("Mode Selection")
    mode_window.geometry("400x250")

    # Create labels and buttons for mode selection
    label = tk.Label(mode_window, text="Select Mode", font=("Arial",
14))
    label.pack(pady=10)
```

### 2.3.6 User Variable Display

This decision was made to display user-specific variables based on the selected mode.

Code:

```
# Function to show the welcome page with user-specific variables
def show_welcome_page(username, mode):
    # Load user variables from JSON file for the selected mode
    user_variables = load_user_variables(username, mode)

    # Create the welcome page window
    global welcome_window
    welcome_window = tk.Tk()
    welcome_window.title("Pacemaker Control Panel")
    welcome_window.geometry("800x400")

    # Create Labels for variables with professional font and colors
    font = ("Arial", 14)
    label_color = "#333"
```

### *2.3.7 Variable Update*

This decision was made to allow users to update variables and save those updates.

Code:

```
def update_variables():
    updated_variables = {}
    for var_name, entry in variable_entries.items():
        updated_value = entry.get()
        updated_variables[var_name] = updated_value

    # Save all updated user variables for the selected mode
    save_user_variables(username, mode, updated_variables)

    messagebox.showinfo("Variables Updated", "User variables updated
successfully.")
```

### *2.3.8 Main User Interface*

This decision was made create a graphical user interface (GUI) for user login and registration.

Code:

```
# Create the main login/registration window
login_window = tk.Tk()
login_window.title("Medical System")
login_window.geometry("400x250")
```

### *2.3.9 GUI Elements*

This decision was made to create labels, entry widgets, and buttons for user interaction.

Code:

```
3   username_label = tk.Label(login_window, text="Username:", font=font,
    fg=label_color, bg="#007acc")
```

```
    username_label.pack(pady=5)
    username_entry = tk.Entry(login_window, font=font, bg=entry_bg_color)
    username_entry.pack(pady=5)

    password_label = tk.Label(login_window, text="Password:", font=font,
    fg=label_color, bg="#007acc")
    password_label.pack(pady=5)
    password_entry = tk.Entry(login_window, show="*", font=font,
    bg=entry_bg_color)
    password_entry.pack(pady=5)

    login_button = tk.Button(login_window, text="Login", command=login,
    font=font, bg="#4CAF50", fg=label_color)
    login_button.pack(pady=10)

    register_button = tk.Button(login_window, text="Register",
    command=register, font=font, bg="#ff3333", fg=label_color)
    register_button.pack(pady=5)
```

### *2.3.10 DCM Connection Status Display*

This decision was made to display the DCM connection status to inform the user.

Code:

```
# Display "DCM connected" message if DCM is connected
if 1:
    dcm_label = tk.Label(login_window, text="DCM connected", font=font,
fg="green")
    dcm_label.pack()
```

These detailed design decisions and corresponding code parts provide a comprehensive overview of the program's structure and functionality, ensuring user authentication, mode-specific variable handling, and a user-friendly graphical interface.

## 2.4 Modules

### Module 1: User Data Management

The purpose of this module manages the loading and saving of user data from a JSON file. It serves as the backend for user authentication and registration processes.

The public functions are as follows:

1. load_users(): Loads user data from the "users.json" file.
    - Parameters: None
2. save_users(data): Saves user data to the "users.json" file.
    - Parameters: data (dictionary containing user data)

The functions with Black-box Behavior are as follows:

1. load_users(): Reads the "users.json" file and returns user data as a dictionary. If the file doesn't exist, it returns an empty dictionary.
2. save_users(data): Writes the provided data to the "users.json" file.

The state variables implemented are:

1. users (dictionary) - Stores user data.

There were no Private Functions included.

The Internal Behavior of the functions:

1. load_users():
    - Internally opens and reads the "users.json" file.
    - Parses the file's contents as JSON and returns it as a dictionary.
    - If the file doesn't exist, returns an empty dictionary.
2. save_users(data):
    - Internally opens the "users.json" file in write mode.
    - Writes the provided data (a dictionary) as a JSON-formatted string to the file.

### Module 2: User Variables Management

The purpose of this module handles loading and saving user-specific variables for each mode (AAI, VVI, VOO, AOO) using JSON files. It manages user-specific settings for different pacemaker modes.

The public functions included in this module is as follows:

1. load_user_variables(username, mode): Loads user-specific variables for a specific mode.
    - Parameters: username (string), mode (string)
2. save_user_variables(username, mode, variables): Saves user-specific variables for a specific mode.
    - Parameters: username (string), mode (string), variables (dictionary of user-specific variables)

The Black-box behavior functions in the module:

1. load_user_variables(username, mode): Reads the <username>_<mode>_variables.json file and returns user variables as a dictionary. If the file doesn't exist, it returns an empty dictionary.
2. save_user_variables(username, mode, variables): Writes the provided variables dictionary to the <username>_<mode>_variables.json file.

There are no state variables or private functions in this module:

The Internal behavior in this module is as follows:

1. load_user_variables(username, mode):
    - Internally opens and reads the <username>_<mode>_variables.json file.
    - Parses the file's contents as JSON and returns it as a dictionary.
    - If the file doesn't exist, returns an empty dictionary.
2. save_user_variables(username, mode, variables):
    - Internally opens the <username>_<mode>_variables.json file in write mode.
    - Writes the provided variables (a dictionary) as a JSON-formatted string to the file.

### Module 3: User Authentication

The purpose of this module authenticates users based on their provided username and password. It handles user login and registration processes.

The public functions included in this module:

1. login(): Handles user login.
    - Parameters: None
2. register(): Handles user registration.
    - Parameters: None

The black-box behavior is as follows:

1. login():
    - Validates the provided username and password against stored user data.
    - If successful, sets current_user to the username and dcm_connected to True.
    - Displays an error message if login fails.
2. register():
    - Checks if the maximum number of users is reached and if the provided username already exists.
    - Creates a new user account if conditions are met, including setting current_user to the new username.
    - Displays an error message if registration fails.

The state variables are as follows:

- users (dictionary) - Stores user data.
- current_user (string) - Tracks the current user.
- dcm_connected (boolean) - Indicates DCM connection status.

There are no private functions for this module.

The internal behavior of the functions includes:

1. login():
    - Internally compares the provided username and password with the stored user data.
    - If the credentials match, it sets current_user and dcm_connected.
    - If login fails, it displays an error message using the messagebox.

2. register():
    - Checks the user limit and existing usernames to determine if registration is allowed.
    - If conditions are met, it creates a new user account by updating the user's dictionary.
    - It sets current_user to the newly registered username.
    - If registration fails, it displays an error message using the messagebox.

### Module 4: Mode Selection

The purpose of this module handles the mode selection process, allowing users to choose between AAI, VVI, VOO, and AOO modes. It initiates the selected mode and moves to the next step.

The public functions of this module include:

1. show_mode_select(username): Displays the mode selection window.
    - Parameters: username (string)

The Black-box behavior of this module is as follows:

1. show_mode_select(username):
    - Displays a window with mode selection buttons (AAI, VVI, VOO, AOO).
    - Selecting a mode initializes user variables for that mode and moves to the welcome page.

There are no state or private variables for this module.

The internal behavior of this module is as follows:

1. show_mode_select(username):
    - Creates a window with mode selection buttons.
    - Initializes user variables for the selected mode (e.g., save_user_variables(username, mode, {})).
    - Proceeds to the welcome page when a mode is selected.

### Module 5: User Variable Display and Update

The purpose of this module displays user-specific variables based on the selected mode and allows users to update these variables. It provides an interface for variable modification and saving changes.

The public functions are as follows:

1. show_welcome_page(username, mode): Displays the welcome page with variable entry fields.
    - Parameters: username (string), mode (string)

The module contains these black-box behaviors:

1. show_welcome_page(username, mode):
    - Displays a window with labels and entry fields for user variables.
    - Allows users to modify variables and save changes.

The state variables of this module is as follows:

- welcome_window - Reference to the welcome page window.
- variable_entries (dictionary) - Tracks user variable entry fields.

The private functions are as follows:

1. update_variables(): Handles variable updates and saves changes.
    - Parameters: None

The Internal behaviors of this module:

1. show_welcome_page(username, mode):
    - Loads user-specific variables for the selected mode from JSON files.
    - Creates a window with labels and entry fields for each variable.
    - Initializes entry fields with existing values.
    - Allows users to modify variable values.
    - Calls update_variables() to save changes and displays a confirmation message.
2. update_variables():
    - Iterates through variable entry fields and retrieves updated values.
    - Saves the updated variables for the selected mode using save_user_variables(username, mode, updated_variables).
    - Displays a confirmation message using the messagebox.

### Module 6: Main Application

The purpose of this module serves as the entry point for the program, creating the main login/registration window and handling user interactions for authentication and mode selection.

There are no public functions for this module.

The black-box behavior in this module:

- Displays the main login/registration window.
- Handles user login, registration, and displays DCM connection status.

There are no state variables or private fucntions in this module.

The Internal behavior of this module:

- Creates the main login/registration window with labels, entry widgets, and buttons.
- Handles user login and registration using login() and register() functions.
- Displays the "DCM connected" message if the DCM is connected.

These detailed descriptions cover the purpose, public functions, black-box behaviors, state variables, private functions, and internal behaviors for each module in the program.

## 2.5 Testing cases

### *Test Case 1: User Registration*

**Test Steps:**

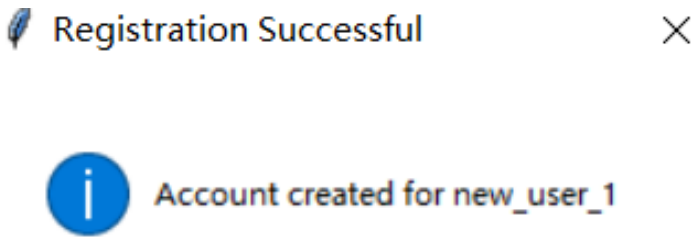
1. Start the application.
2. Click the "Register" button.
3. Enter a unique username (e.g., "new_user_1") and a password (e.g., "password123").
4. Click the "Register" button.

**Expected Results:**

- A successful registration message should appear.
- The user should be able to log in using the newly registered credentials.

Actual Results:

- A message indicates successful registration appeared on the screen.



*Test Case 2: Duplicate Username Registration*

**Test Steps:**

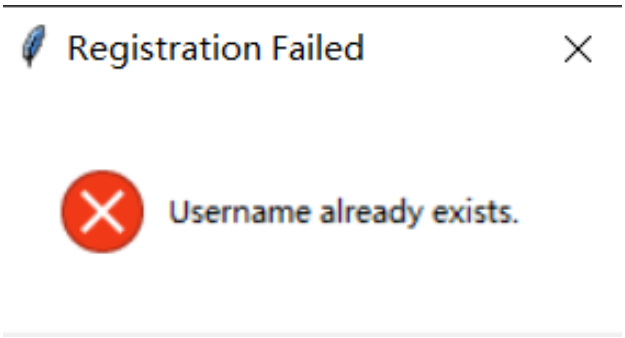
1. Start the application.
2. Click the "Register" button.
3. Enter an existing username (e.g., "user1") and a password (e.g., "password123").
4. Click the "Register" button.

**Expected Results:**

- An error message should appear, indicating that the username already exists.
- The user should not be registered with the same username.

Actual results:

- The error message appears on the screen.



*Test Case 3: Blank Username or Password*

**Test Steps:**

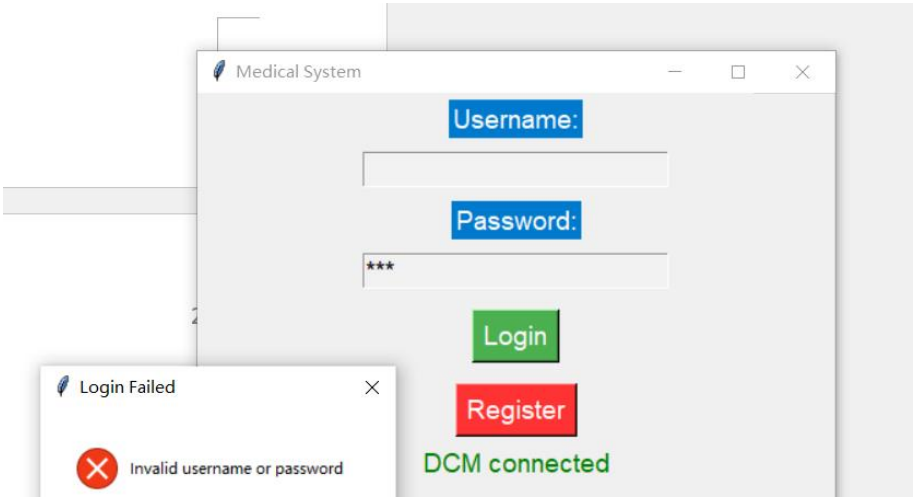
1. Start the application.
2. Click the "Register" button.
3. Leave the username field blank and enter a password (e.g., "password123").
4. Click the "Register" button.

**Expected Results:**

- An error message should appear, indicating that the username and password are required.
- The user should not be registered with blank fields.

Actual results:

- An error message appeared.



### *Test Case 4: Invalid Login*

**Test Steps:**

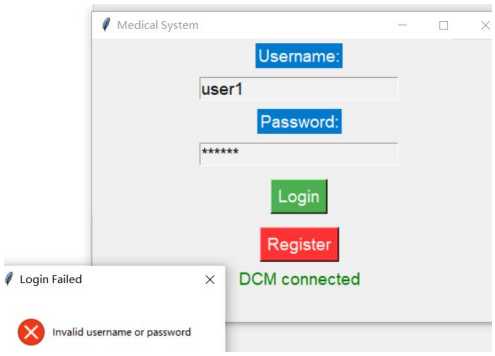
1. Start the application.
2. Click the "Login" button.
3. Enter a valid username (e.g., "user1") and an incorrect password (e.g., "wrong_password").
4. Click the "Login" button.

**Expected Results:**

- An error message should appear, indicating that the login failed.
- The user should not be logged in with an incorrect password.

Actual results:

- An error message appeared.



*Test Case 5: Mode Selection*

**Test Steps:**

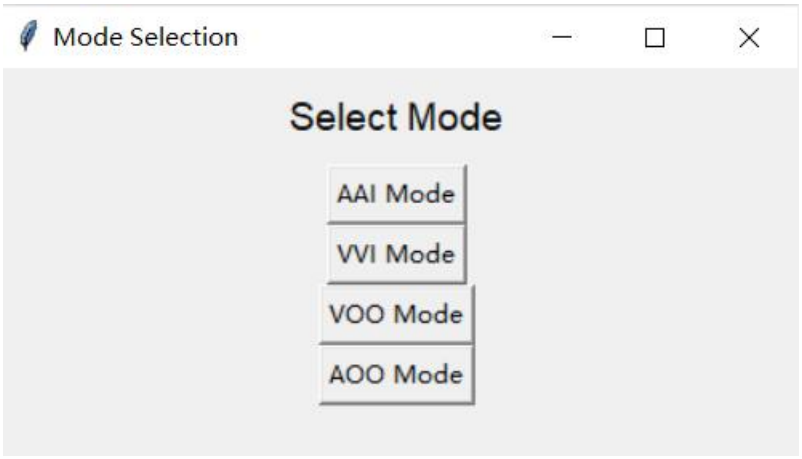
1. Start the application.
2. Register a new user (e.g., "new_user_2").
3. Select the "AAI" mode.
4. Click the "Back to Mode Selection" button.
5. Select the "VVI" mode.
6. Click the "Back to Mode Selection" button.
7. Select the "VOO" mode.
8. Click the "Back to Mode Selection" button.
9. Select the "AOO" mode.

**Expected Results:**

- The user should be able to switch between modes and return to the mode selection screen without issues.

Actual results:

- The program allows the users to switch between 4 operating modes.



*Test Case 6: Variable Update Flow*

**Test Steps:**

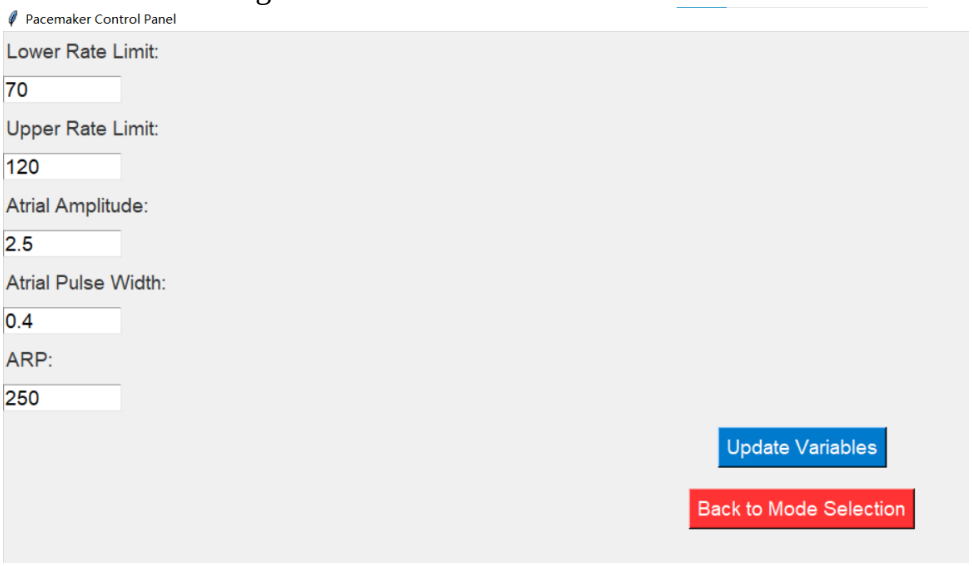
1. Start the application.
2. Register a new user (e.g., "new_user_3") and select the "AAI" mode.
3. Update variable values:
    - Lower Rate Limit: 70
    - Upper Rate Limit: 120
    - Atrial Amplitude: 2.5
    - Atrial Pulse Width: 0.4
    - ARP: 250
4. Click the "Update Variables" button.
5. Log out.
6. Log in with the same user ("new_user_3") and select the "AAI" mode.

**Expected Results:**

- Variable updates should be preserved between sessions.
- Logging in with the same user and mode should display the updated values (e.g., Lower Rate Limit: 70, Upper Rate Limit: 120, Atrial Amplitude: 2.5, Atrial Pulse Width: 0.4, ARP: 250).

Actual results:

- The updated variables along with its values are shown:



### *Test Case 7: Mode-Specific Variables*

**Test Steps:**

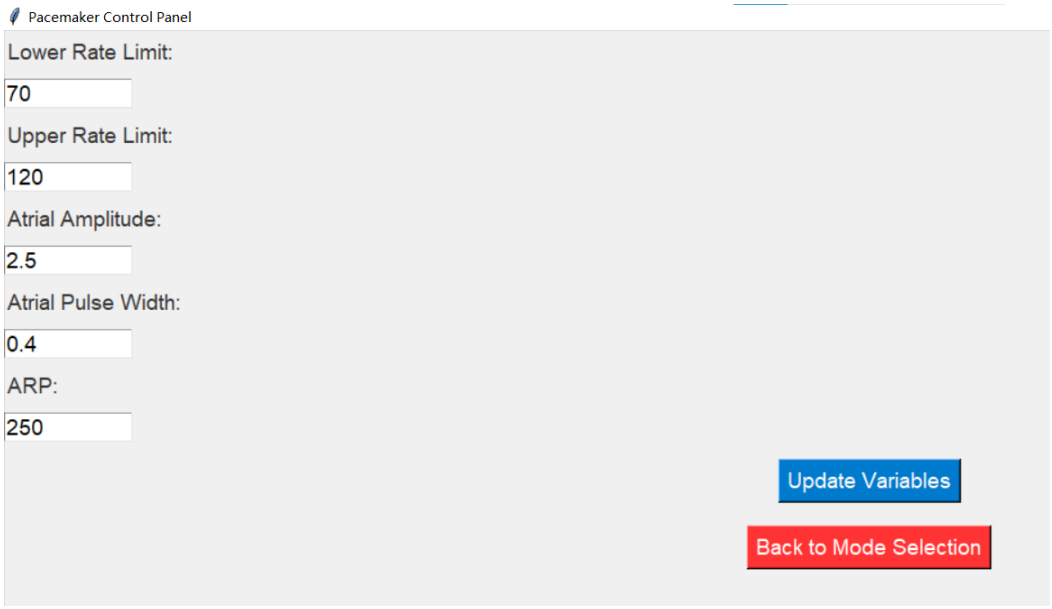
1. Start the application.
2. Register a new user (e.g., "new_user_4") and select the "AAI" mode.
3. Update variable values:
    - Lower Rate Limit: 70
    - Upper Rate Limit: 120
    - Atrial Amplitude: 2.5
    - Atrial Pulse Width: 0.4
    - ARP: 250
4. Click the "Update Variables" button.
5. Select the "VVI" mode and update variables specific to VVI mode:
    - Lower Rate Limit: 60
    - Upper Rate Limit: 130
    - Ventricular Amplitude: 3.0
6. Click the "Update Variables" button.
7. Log out.
8. Log in with the same user ("new_user_4") and select the "AAI" mode.

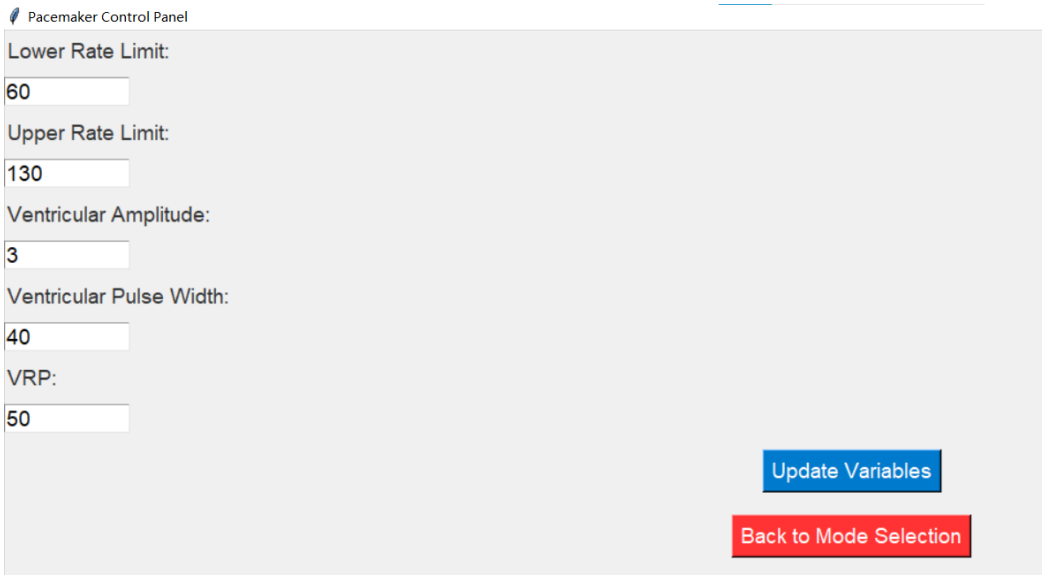
**Expected Results:**

- Variable updates should be mode-specific, and changing variables in one mode should not affect variables in another mode.
- Logging in with the same user in a different mode should display the mode-specific variables.

Actual results:

- The variables are updated according to its operating mode, even thought they have the same variable name, their values are independent from each other.


Pacemaker Control Panel
Lower Rate Limit:
70
Upper Rate Limit:
120
Atrial Amplitude:
2.5
Atrial Pulse Width:
0.4
ARP:
250
Update Variables
Back to Mode Selection


Pacemaker Control Panel
Lower Rate Limit:
60
Upper Rate Limit:
130
Ventricular Amplitude:
3
Ventricular Pulse Width:
40
VRP:
50
Update Variables
Back to Mode Selection

*Test Case 8: Invalid Variable Values*

**Test Steps:**

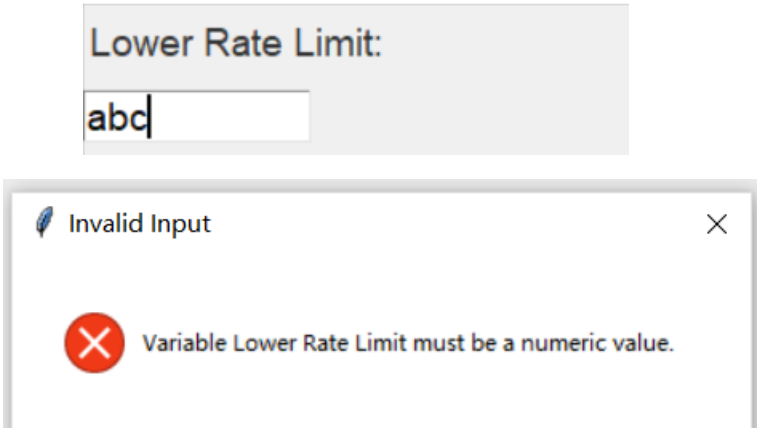
1. Start the application.
2. Register a new user (e.g., "new_user_5") and select the "AOO" mode.
3. Attempt to update variable values with invalid input, such as non-numeric characters or values outside valid ranges.
    - Set Lower Rate Limit to "abc."
    - Set Upper Rate Limit to "-10."
    - Set Atrial Amplitude to "2.5.1"
    - Set Atrial Pulse Width to "0.0"
    - Set ARP to "200.5"

**Expected Results:**

- An error message should appear for each invalid input.
- The user should not be able to update variables with invalid values.

Actual results:

- The error message occurred when the user input is not a numeric value.




These detailed test cases cover various scenarios, including registration, login, mode selection, variable updates, mode-specific variables, and handling of invalid inputs.

## References

[1] McMaster University. (Fall 2023). PACEMAKER System Specification. [Online]. Available: https://avenue.cllmcmaster.ca/d2l/le/content/557175/viewContent/4323499/View?ou=557175

[2] McMaster University. (Fall 2023). What is a Pacemaker? [Online]. Available: https://avenue.cllmcmaster.ca/d2l/le/content/557175/viewContent/4324589/View