Deliverable 1: Documentation

Group 9

Fall 2025

Contents

1 Group Members

Yutong Liu, liu1828, 400492297

Daniel Kim, kim600, 400505173

Alex Melnbardis, melnbaa, 400447130

Farhan Sifar, sifarf, 400515680

Samarth Vijay, vijays28, 400512489

Ray Yu, yu321, 400505288

2 Part 1

2.1 Introduction

Purpose: We implement a bradycardia pacemaker using model-driven design (Simulink/Stateflow) and a desktop Device Controller–Monitor (DCM). The embedded controller senses atrial/ventricular events, enforces timing cycles, and emits safe pacing pulses.
D1 Scope: Deliver a documented, annotated model for AOO, VOO, AAI, VVI, and a working DCM front-end (accounts, parameter entry, validation, profiles). Serial/UART integration and hardware telemetry are part of a later deliverable.
Goals of this part. (i) Present formal, disjoint, traceable requirements; (ii) justify design decisions; (iii) include Simulink screenshots with annotations; (iv) include DCM screenshots; (v) provide testing performed and results; (vi) present validation & verification for both DCM and pacemaker.

2.2 Requirements

2.2.1 System Requirements

- SR-1 (Supported modes). The pacemaker shall implement AOO, VOO, AAI, VVI, selected via ModeSelect (enumeration agreed with the DCM).
- SR-2 (Timing parameters). The controller shall accept LRL_interval_ms (low-rate interval), ARP_ms (atrial refractory), and VRP_ms (ventricular refractory) as non-negative integer milliseconds.

- SR-3 (Sensing inputs). ATR_CMP_DETECT (bool) and VENT_CMP_DETECT (bool) shall represent chamber-specific sensed events after comparator/thresholding.
- SR-4 (Pacing outputs). o_ap and o_vp shall denote atrial/ventricular pace events; they are either one-tick triggers to a Pulse-Width Generator (preferred in D1) or held high for the programmed pulse width.
- SR-5 (Timing-cycle semantics). The controller shall enforce the timing cycle with automatic interval LRL_interval_ms, demand inhibition (AAI/VVI), and chamber-specific refractory lockouts lasting ARP_ms/VRP_ms.
- SR-6 (Interface contract). All ports and units shall match the Simulink block signature shown in §2.3.

2.2.2 Mode Requirements

- MR-AOO. Ignore both senses; produce atrial paces periodically at LRL_interval_ms.
- MR-VOO. Ignore both senses; produce ventricular paces periodically at LRL_interval_ms.
- MR-AAI. In AAI, if ATR_CMP_DETECT occurs before LRL_interval_ms and outside ARP_ms, inhibit the pending atrial pace and restart the interval; otherwise, pace atrium at timeout and start ARP_ms.
- MR-VVI. In VVI, if VENT_CMP_DETECT occurs before LRL_interval_ms and outside VRP_ms, inhibit the pending ventricular pace and restart the interval; otherwise, pace the ventricle at timeout and start VRP_ms.

2.2.3 DCM Requirements

- DCM-1 (Users & profiles). Provide local login/registration (≤10 users) and per-user profiles; maintain a reserved __last__ profile updated on Apply.
- DCM-2 (Parameters & validation). Surface Mode, LRL (ppm) with computed LRL_interval_ms, ARP_ms, VRP_ms, and A/V Amplitude/PulseWidth fields. Enforce UI-level constraints in D1:
    - LRL 30–175 ppm → LRL_interval_ms = round(60000 / LRL)
    - ARP_ms, VRP_ms ∈ [100, 500]
    - Amplitudes ∈ [0.1, 5.0] V; PulseWidths ∈ [0.1, 30] ms
    - Mode ∈ {AOO, VOO, AAI, VVI}
- DCM-3 (Scope disclosure). D1 is front-end only: no UART/telemetry yet; that is planned for the next deliverable.

2.3 Design

2.3.1 System Architecture

- DCM (host): Tkinter app (Modes / Parameters / Profiles / About). JSON-backed, validated inputs. Displays LRL and computed LRL interval (60000/LOWER_RATE_LIMIT).
- PacemakerController (model): One Simulink/Stateflow chart with four mode substates (AOO, VOO, AAI, VVI). Demand modes optionally enter a Hysteresis substate.
- Interface layer: Digital reads for atrial/ventricular senses; PWM for pacing amplitude and sense thresholds; digital writes for charge/ground and pace control.

2.3.2 Programmable Parameters

- Timing: LOWER_RATE_LIMIT (ppm), UPPER_RATE_LIMIT (ppm), ATR_PULSE_WIDTH (ms), VENT_PULSE_WIDTH (ms), ATR_REFRAC_PERIOD (ms), VENT_REFRAC_PERIOD (ms), HYSTERESIS (bool), HYSTERESIS_INTERVAL (ms).
- Amplitude & thresholds (mapped to PWM duty cycle): ATR_PULSE_AMP_REG, VENT_PULSE_AMP_REG, ATR_SENS, VENT_SENS.
- Mode: MODE_SELECT ∈ {AOO=0, VOO=1, AAI=2, VVI=3}.

2.3.3 Hardware Inputs/Outputs

- Inputs: ATR_CMP_DETECT, VENT_CMP_DETECT, MODE_SELECT, all parameters above.
- Outputs:
    - Pacing: ATR_PACE_CTRL, VENT_PACE_CTRL.
    - PWM: PACER_REF_PWM (pace amplitude), ATR_CMP_REF_PWM, VENT_CMP_REF_PWM (sense thresholds).
    - Support: PACE_CHARGE_CTRL, PACE_GND_CTRL, ATR_GND_CTRL, VENT_GND_CTRL, Z_ATR_CTRL, Z_VENT_CTRL, FRONTEND_CTRL.

2.3.4 State Machines (per mode)

- Common skeleton: Charging_and_Sensing → (Hysteresis in demand modes when enabled) → Pacing → back to Charging_and_Sensing. Temporal counters implement delays and pulse widths.
- AOO / VOO (asynchronous): Pace at programmed rate; pulse width sets pacing duration.
- AAI / VVI (demand): If no valid sense occurs after pulse width + refractory within the LRL interval, pace. A valid sense (outside refractory) resets the timer; if HYSTERESIS is true, the next interval is extended by HYSTERESIS_INTERVAL.
- Refractory: ATR/VENT_REFRAC_PERIOD blocks sensing during/after a pulse.
- PWM roles: PACER_REF_PWM sets pacing energy; ATR_CMP_REF_PWM/VENT_CMP_REF_PWM set independent sensing thresholds.

2.3.5 Simulink Diagram

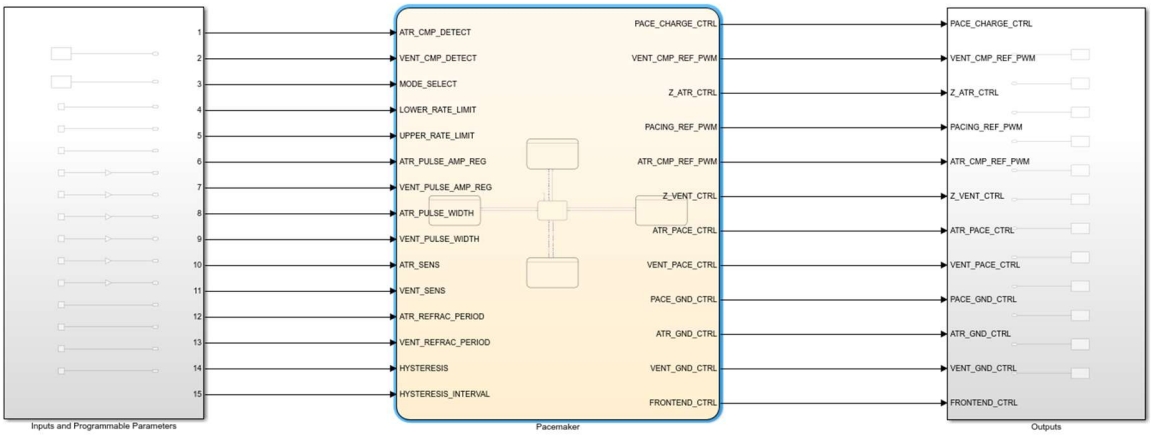


Figure 1: Top level Simulink model

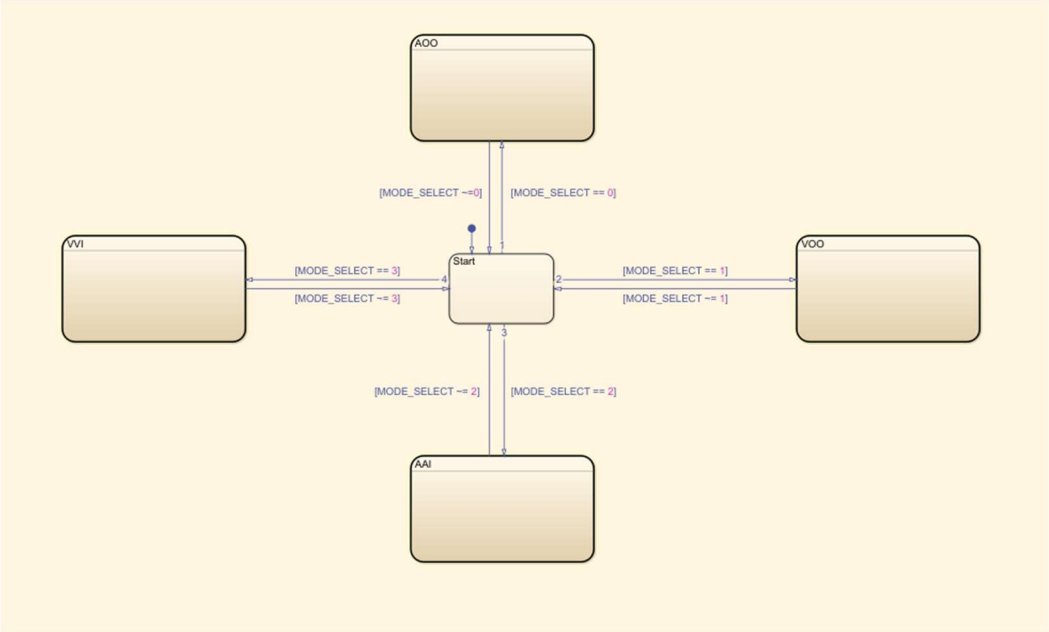


Figure 2: Mode selector subsystem

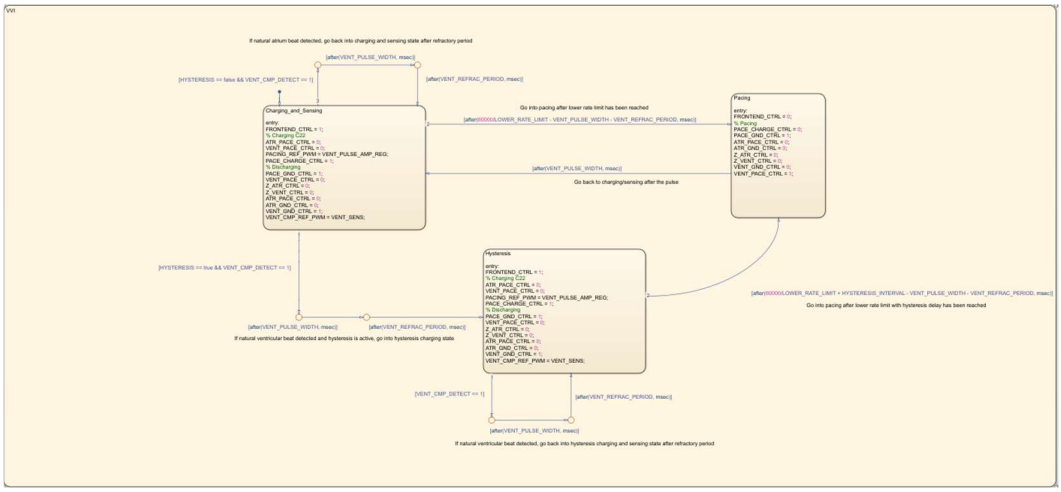


Figure 3: VVI mode FSM

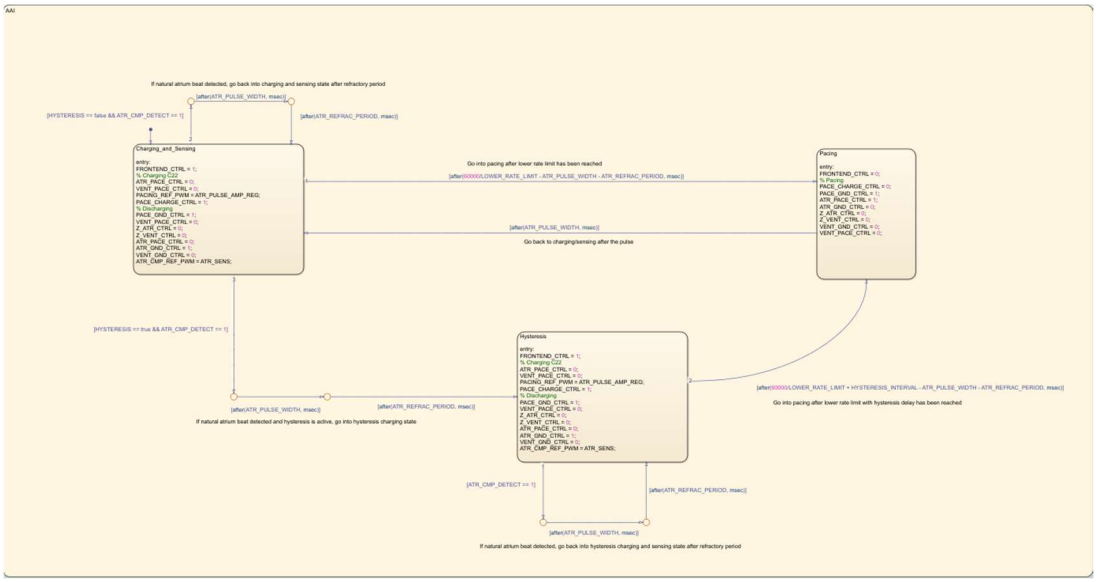


Figure 4: AAI mode FSM

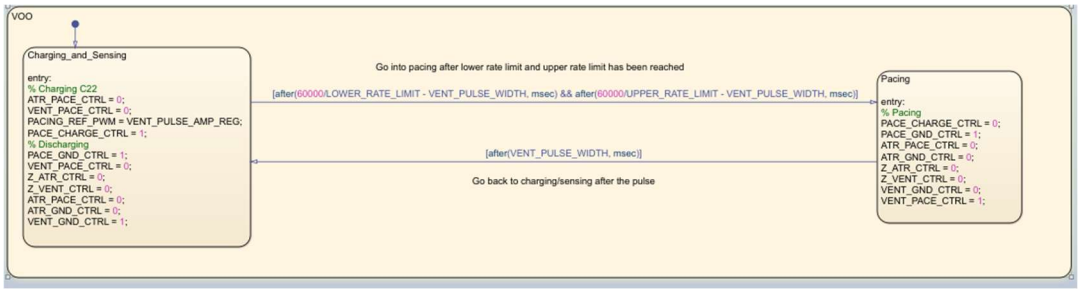


Figure 5: VOO mode FSM

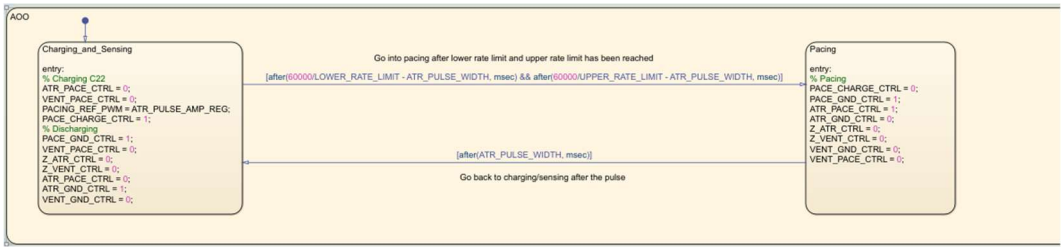


Figure 6: AOO mode FSM

2.3.6 DCM Screenshots

The GUI python files are split into 6 python files, with the main UI python file named app.py. The sub/header files are imported into the main file and each control and direct a class for a particular part of the UI.

```python
from __future__ import annotations
import tkinter as tk
from tkinter import ttk, messagebox

class ParametersView(ttk.Frame):
    RANGES = {
        "LRL": "[30–175 ppm]",
        "URL": "(> LRL) [up to 220 ppm]",
        "AtrialAmplitude": "[0.1–5.0 V]",
        "AtrialPulseWidth": "[0.1–30 ms]",
        "VentricularAmplitude": "[0.1–5.0 V]",
        "VentricularPulseWidth": "[0.1–30 ms]",
        "ARP": "[100–500 ms]",
        "VRP": "[100–500 ms]",
    }

    def __init__(self, parent, app):
        super().__init__(parent, padding=12)
        self.app = app
        self.param_vars: dict[str, tk.StringVar] = {}
        self.mode_var = tk.StringVar(value=self.app.current_params.get("Mode", "AOO"))
        self._build()

    def _build(self):
        grid = ttk.Frame(self)
        grid.pack(fill=tk.X)

        fields = [
            ("LRL (ppm)", "LRL"),
            ("URL (ppm)", "URL"),
            ("Atrial Amplitude (V)", "AtrialAmplitude"),
            ("Atrial Pulse Width (ms)", "AtrialPulseWidth"),
            ("Ventricular Amplitude (V)", "VentricularAmplitude"),
            ("Ventricular Pulse Width (ms)", "VentricularPulseWidth"),
            ("ARP (ms)", "ARP"),
```

Figure 7: DCM GUI overview and module layout

The different parts are; dcm_visual_view, modes_view, parameters_view, profiles_view, registration_view, and about_view.

```python
from __future__ import annotations
import tkinter as tk
from tkinter import ttk, messagebox

class ModesView(ttk.Frame):
    def __init__(self, parent, app):
        super().__init__(parent, padding=12)
        self.app = app
        self.mode_var = tk.StringVar(value=self.app.current_params.get("Mode", "AOO"))
        self._build()

    def _build(self):
        ttk.Label(self, text="Select Pacing Mode", font=("Segoe UI", 12, "bold")).pack(anchor="w")
        options = [
            ("AOO – Atrial Asynchronous", "AOO"),
            ("VOO – Ventricular Asynchronous", "VOO"),
            ("AAI – Atrial Demand", "AAI"),
            ("VVI – Ventricular Demand", "VVI"),
        ]
        grp = ttk.Frame(self)
        grp.pack(anchor="w", pady=(6,12))
        for i, (label, val) in enumerate(options):
            ttk.Radiobutton(grp, text=label, value=val, variable=self.mode_var, command=self._on_mode_change).grid(row=i, column=0, sticky

    def on_session_start(self):
        self._refresh_mode_ui()

    def _refresh_mode_ui(self):
        self.mode_var.set(self.app.current_params.get("Mode", "AOO"))

```

Figure 8: GUI components

In the main app.py file, there is a class called DCM_STORE which allows all information through the UI to be stored in a json config file called dcm_config.json.

```python
class DCMStore:
    #Simple JSON file based storage for users and individual settings
    def __init__(self, path: str):
        self.path = path
        if not os.path.exists(path):
            self._init_file()
        self._load()

    def _init_file(self):
        payload = {"users": {}, "profiles": {}}
        with open(self.path, "w", encoding="utf-8") as f:
            json.dump(payload, f, indent=2)

    def _load(self):
        with open(self.path, "r", encoding="utf-8") as f:
            self.data = json.load(f)
        self.data.setdefault("users", {})
        self.data.setdefault("profiles", {})

    def _save(self):
        with open(self.path, "w", encoding="utf-8") as f:
            json.dump(self.data, f, indent=2)

    # User Functions
    def user_count(self) -> int:
        return len(self.data.get("users", {}))

    def has_user(self, username: str) -> bool:
        return username in self.data.get("users", {})

    def add_user(self, username: str, password: str) -> bool:
        if self.user_count() >= MAX_USERS:
            return False
```

Figure 9: DCM_STORE in app.py

This saves all information related to profiles, modes, users and hash SHA256 encoded passwords for safety.

Figure 10: Config contents

In terms of further deliverables, we created a tab for DCM visuals that are tied currently to Boolean variables to simulate the idea of having other devices nearby and detecting connected devices with serial UART. These can be seen in the code for dcm_visuals_view.py.

```
61
62          def _test_connect(self):
63              #Test button: Connect device
64              self.app.device_com = True
65
66          def _test_disconnect(self):
67              #Test button: Disconnect device
68              self.app.device_com = False
69
70          def _test_detect_pacemaker(self):
71              #Test button: Detect other pacemaker
72              self.app.other_pacemaker_detected = True
73
74          def _test_clear_pacemaker(self):
75              #Test button: Clear other pacemaker
76              self.app.other_pacemaker_detected = False
77
78          def on_session_start(self):
79              #Called when user logs in
80              self._refresh_status()
81
```

Figure 11: DCM Device Visuals tab simulating UART device detection

Using Tkinter for Python GUI and functions of the library as well as helper / test functions in each sub class, we were able to create an interactive functional UI for the DCM.

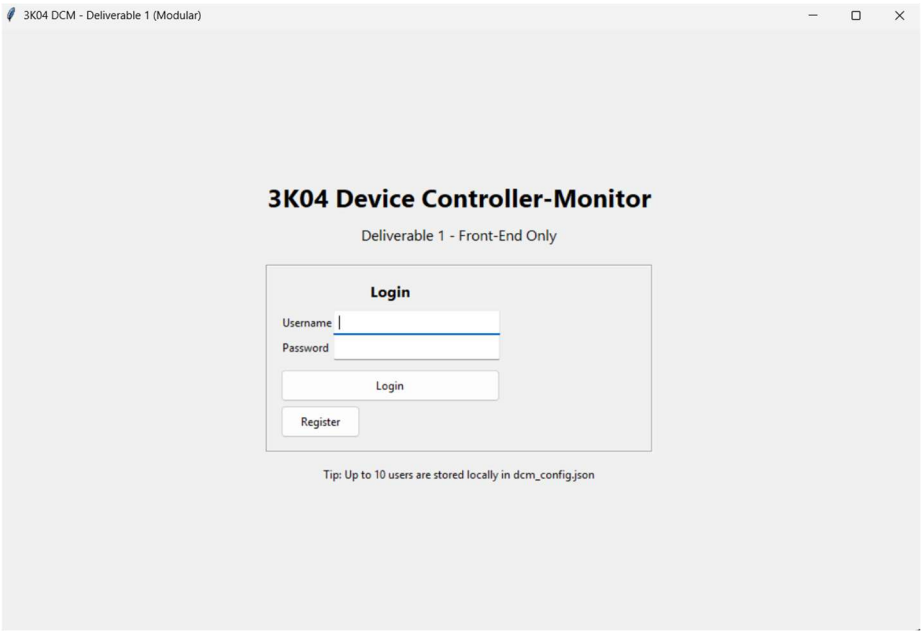


Figure 12: Login screen for DCM

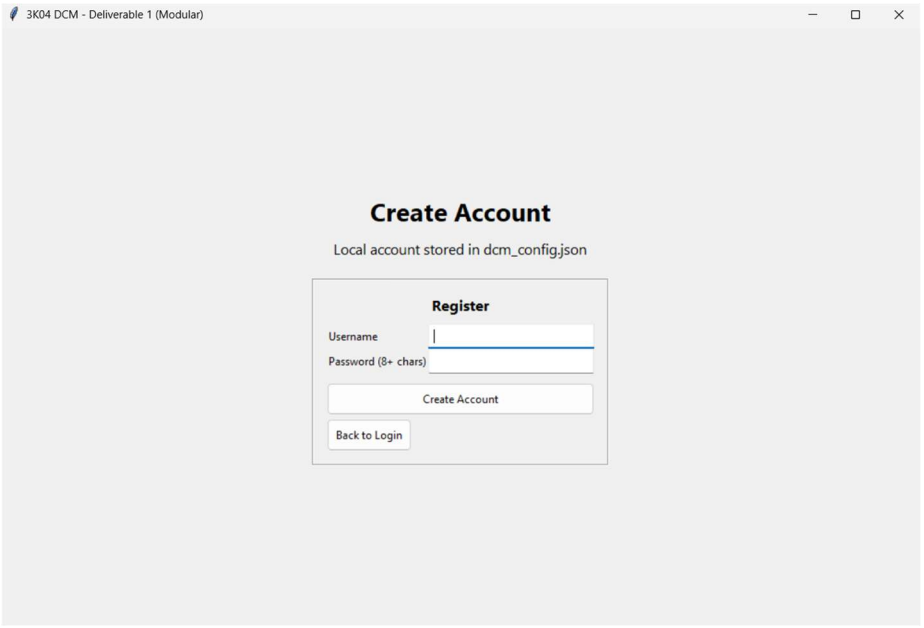


Figure 13: Register account screen for DCM

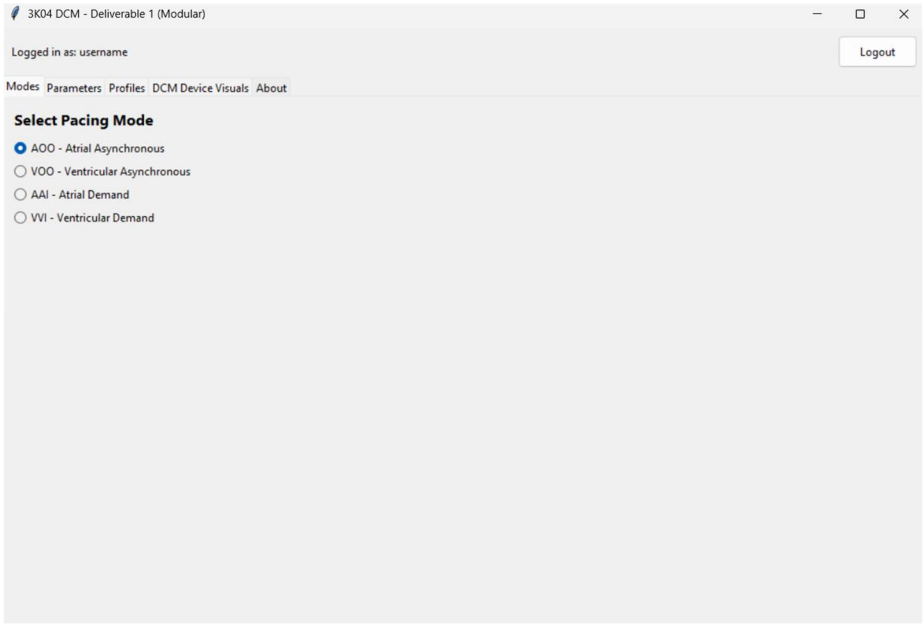


Figure 14: Modes tab

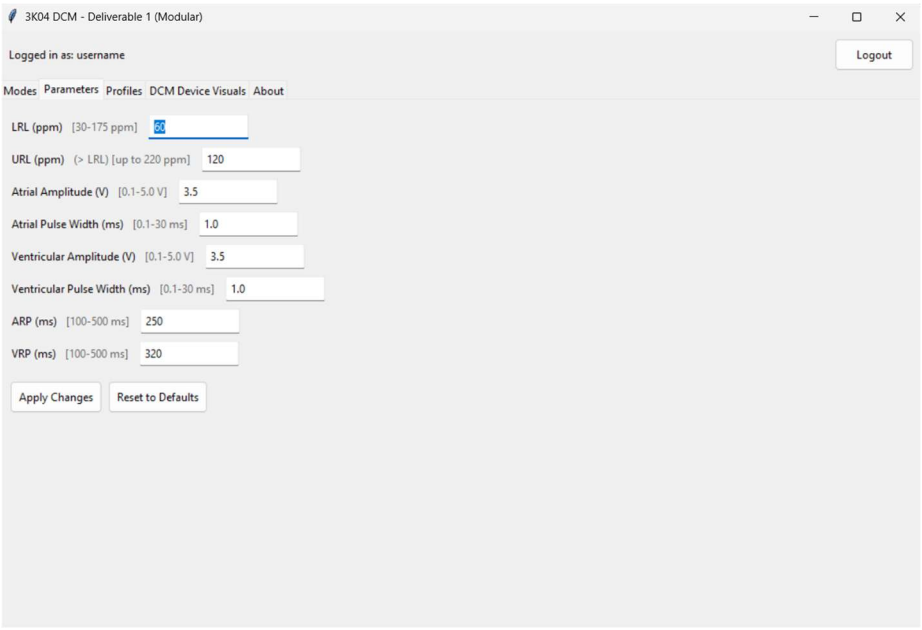


Figure 15: Parameters tab

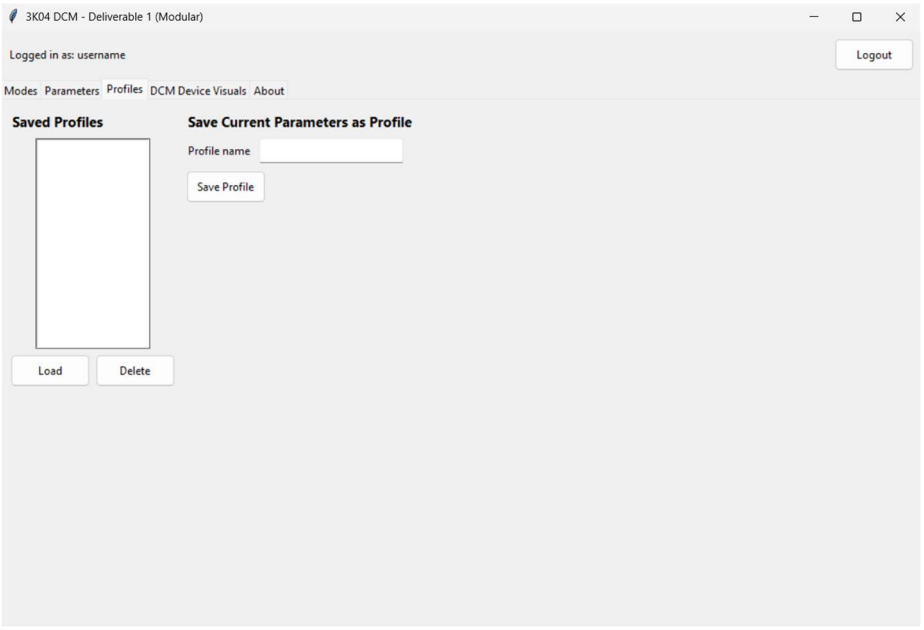


Figure 16: Profiles tab

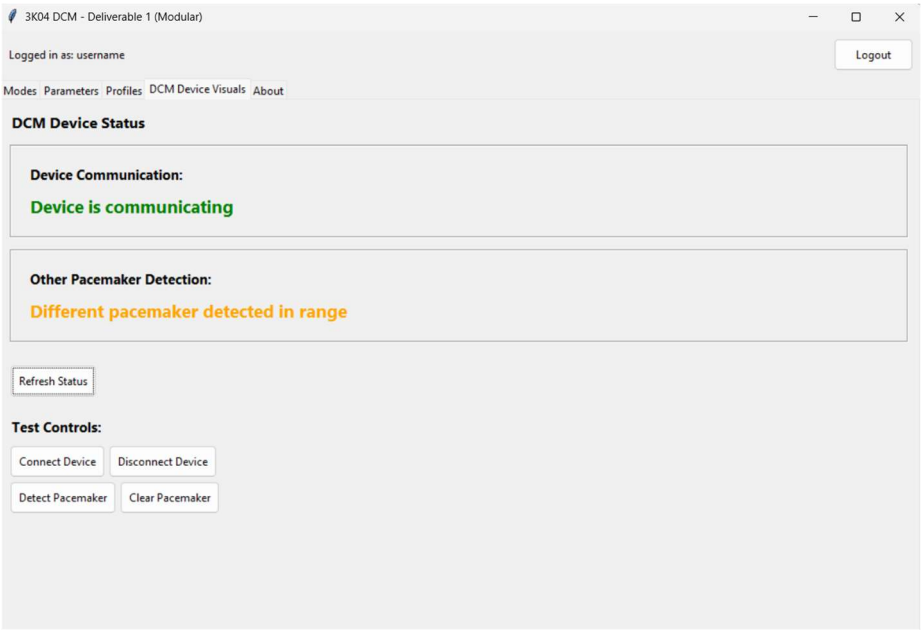


Figure 17: DCM Device Visuals tab

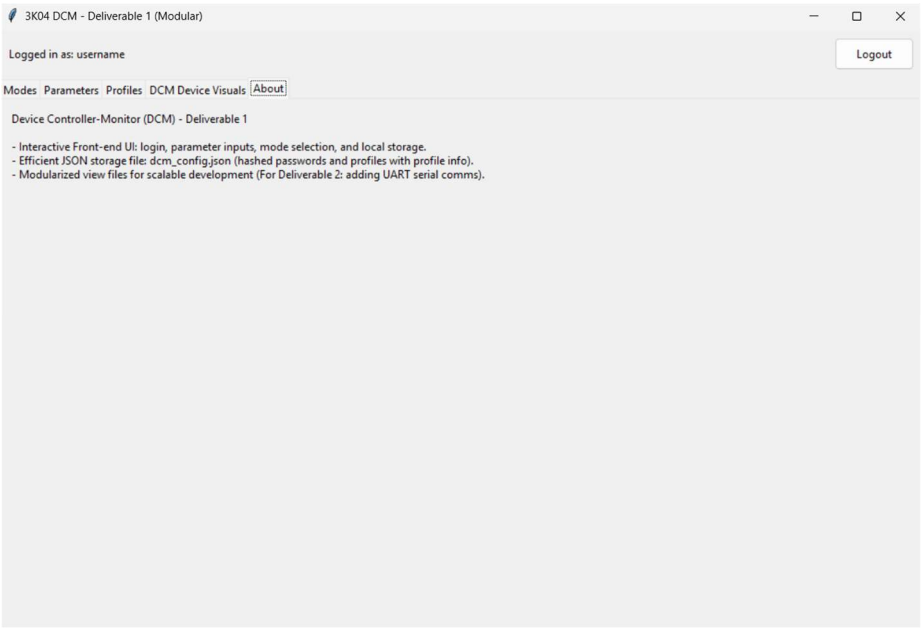


Figure 18: About tab

3 Part 2

3.1 Requirements Potential Changes

- UART/Telemetry**:** Add serial transport, parameter download, and streaming indicators/egrams.
- Parameter tolerances**:** Tighten amplitude/width/rate ranges to hardware-faithful limits.
- Optional features: Rate-adaptive logic; additional safety timers.

3.2 Design Decision Potential Changes

- Packet protocol: Function codes, frame format, checksum/CRC policy.
- UI telemetry: Live traces, comms status panes, printable reports.
- Chart refactoring**:** Shared subcharts/utilities across modes once dual-chamber/RA features are considered.

3.3 Module Description

3.3.1 Module: PacemakerController (Simulink Subsystem)

(a) Purpose: Execute bradycardia timing cycles (AOO/VOO/AAI/VVI) and produce chamber-specific pacing events.
(b) Public interface: ATR_CMP_DETECT: bool, VENT_CMP_DETECT: bool, ModeSelect: uint8, LRL_interval_ms: uint16, ARP_ms: uint16, VRP_ms: uint16 → o_ap: bool, o_vp: bool.
(c) Black-box behaviour: AOO/VOO: periodic pacing; AAI/VVI: demand inhibit + refractory lockout.
(d) State variables: inARP: bool, inVRP: bool, t_cycle_start: time stamp.
(e) Private functions: startRefractory(chamber, dur_ms), validSense(chamber); pulse request to the Pulse-Width Generator.
(f) Internal behaviour:

- WAIT: start/continue interval; if after(LRL_interval_ms, msec) → PACE; if demand sense and validSense → inhibit and t_cycle_start ← now.
- PACE: assert o_ap or o_vp (one-tick trigger); call startRefractory.
- REFRACTORY: mask senses for ARP_ms/VRP_ms; auto-transition to WAIT.

3.3.2 Module: AOOChart / VOOChart (Stateflow)

(a) Purpose: Asynchronous pacing.
(b) Public: uses the timing inputs; no sense inputs.
(c) Black-box: pace at LRL_interval_ms regardless of senses.
(d) State vars: t_cycle_start.
(e) Private: none required.
(f) Internals: single loop WAIT → PACE → WAIT, with after(LRL_interval_ms, msec).

3.3.3 Module: AAIChart (Stateflow)

(a) Purpose: Demand atrial pacing with ARP lockout.
(b) Public: uses ATR_CMP_DETECT, LRL_interval_ms, ARP_ms.
(c) Black-box: inhibit if valid AS; otherwise pace at interval; start ARP after pace.
(d) State vars: inARP, t_cycle_start.
(e) Private: startRefractory(ATR, ARP_ms), validSense(ATR)=!inARP.
(f) Internals: WAIT (sense inhibits) → PACE (emit o_ap) → ARP (mask senses) → WAIT.

3.3.4 Module: VVIChart (Stateflow)

(a) Purpose: Demand ventricular pacing with VRP lockout.
(b) Public: uses VENT_CMP_DETECT, LRL_interval_ms, VRP_ms.
(c) Black-box: inhibit if valid VS; otherwise pace at interval; start VRP after pace.
(d) State vars: inVRP, t_cycle_start.
(e) Private: startRefractory(VENT, VRP_ms), validSense(VENT)=!inVRP.
(f) Internals: WAIT (sense inhibits) → PACE (emit o_vp) → VRP (mask senses) → WAIT.

3.3.5 Module: PulseWidthGenerator (Simulink Subsystem)

(a) Purpose: Shape one-tick triggers into pulses of A/V programmable widths and amplitudes.
(b) Public: ap_trig, vp_trig, A_PW_ms, V_PW_ms, A_Amp_V, V_Amp_V → o_ap, o_vp.
(c) Black-box: for each trigger, assert output for programmed duration at nominal amplitude (logical driver in D1; analog driver in later deliverable).
(d) State vars: ap_end_time, vp_end_time.
(e) Private: none.
(f) Internals: on rising edge, set *_end_time = now + *_PW_ms; output high while now < *_end_time.

3.3.6 Module: DCMStore (Data Layer)

(a) Purpose: Persist users and profiles in a JSON file.
(b) Public: add_user(u,p), verify_user(u,p), save_profile(u,name,params), load_profile(u,name), list_profiles(u).
(c) Black-box: ≤10 users; per-user map of profiles; reserved __last__ auto-updated on Apply.
(d) State vars: data = {users:{u→hash}, profiles:{u→{name→params}}}, path.
(e) Private: _load(), _save(), _init_schema().
(f) Internals: initialize schema if missing, hash passwords, enforce limits, flush to disk on save.

3.3.7 Module: Validator (Pure Function)

(a) Purpose: Enforce UI-level ranges and relationships.
(b) Public: validate(params)->(ok:bool, msg:str).
(c) Black-box: returns (False, reason) if any check fails; else (True,"OK").
(d) State vars: none.

(e) Private: none.
(f) Internals: check each field (ranges above); compute LRL_interval_ms from LRL (ppm); confirm Mode ∈ set.

3.3.8 Module: DCMApp (UI Shell)

(a) Purpose: Provide tabs (Modes/Parameters/Profiles/About), login/registration, and session management.
(b) Public: login_success(username), logout(), show(frame_name).
(c) Black-box: routes frames; propagates session to dashboards.
(d) State vars: active_user, frames, store.
(e) Private: _build(), _style().
(f) Internals: construct frames; wire buttons (Apply→validate→store→__last__).

3.4 Testing

3.4.1 DCM Tests

Registration and Login Test (registration_views.py, app.py)

1. **Purpose**
   To verify that user registration and login functionalities operate correctly, ensuring secure password hashing, proper validation, and appropriate error feedback.
2. **Input conditions**
   New username: testuser5
   Password: strongpass123 (13 characters)
   Attempt registration, then login with the same credentials.
   Attempt login with incorrect password wrongpass123.
   Attempt registering an existing username.
3. **Expected output**
   - Registration displays dialog: “Account created” with text “Registration successful. You can now log in.”
   - JSON entry created under users with a SHA-256 hash and an empty profile dictionary.
   - Valid login transitions directly to the main dashboard without error.
   - Invalid login triggers “Login failed” dialog: “Invalid username or password.”
   - Duplicate username triggers “Exists” dialog: “That username is already taken.”
4. **Actual output**

- Successful registration displayed:

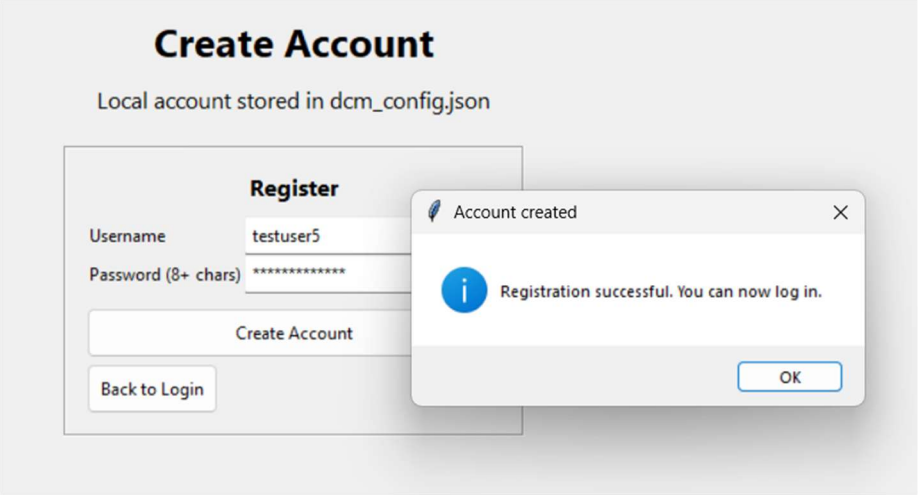


Figure 19: Successful account creation dialog

- dcm_config.json updated as:

```
"testuser5": {
  "password_hash": "378da19515a2c49ba3c45e7e82fafc978c1519b43410deb4673eee02f59d0658"
}
```

Figure 20: dcm_config.json updated after registration

- Valid login displayed no popup and transitioned to dashboard.

- Invalid login displayed:

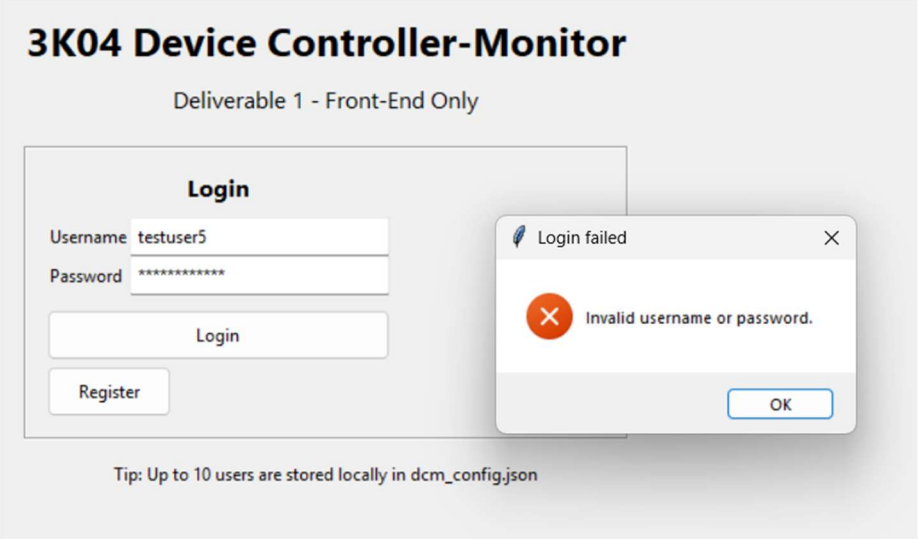


Figure 21: Invalid login error

- Re-registration attempt displayed:

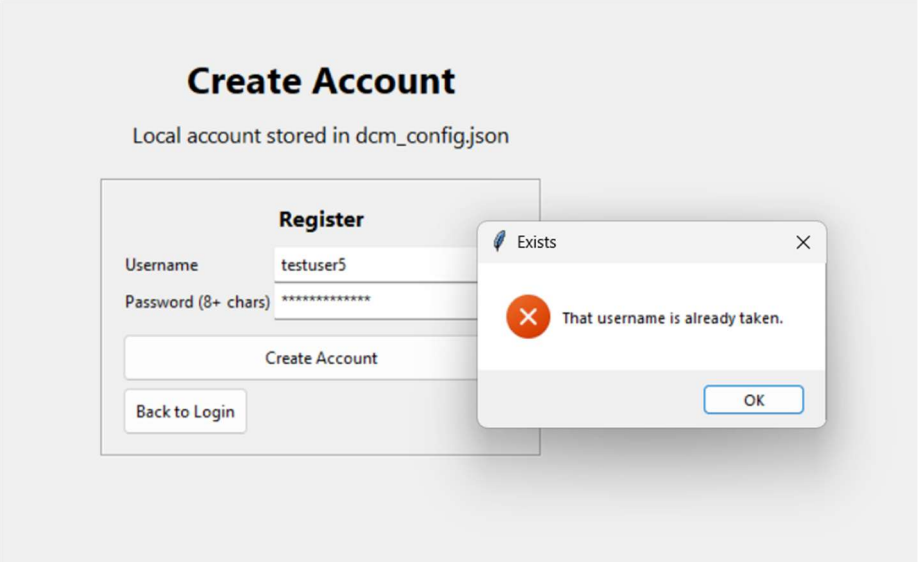


Figure 22: Duplicate username message

5. Result

Pass. Registration, hashing, and login validation behaved correctly and reflected secure JSON updates.

Mode Selection Test (modes_view.py)

1. **Purpose**
   To confirm that the pacing mode selection feature updates the current session parameters.
2. **Input conditions**
   Logged in as testuser5.
   Select AOO, AAI, and VVI modes in sequence using radio buttons.
3. **Expected output**
   - Each selection displays a dialog titled “Mode updated” with text “Mode set to <mode>”.
4. **Actual output**
   - Dialog sequence:

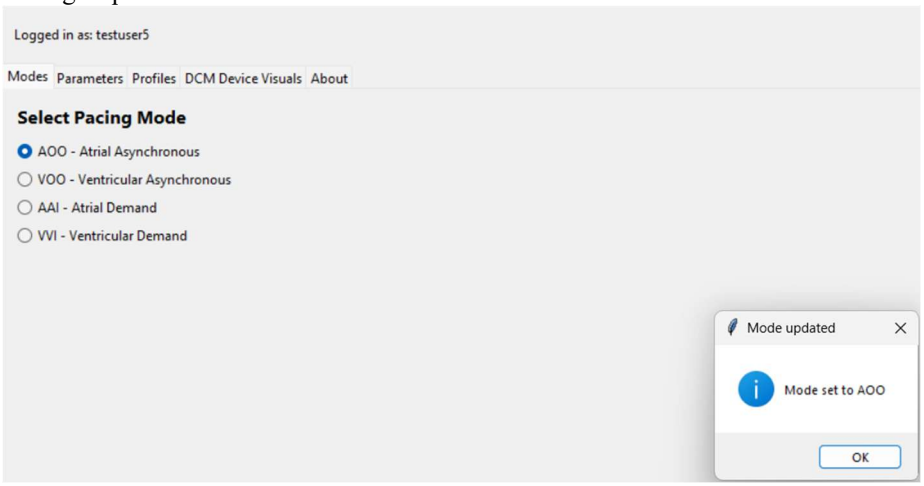


Figure 23: Mode set to AOO

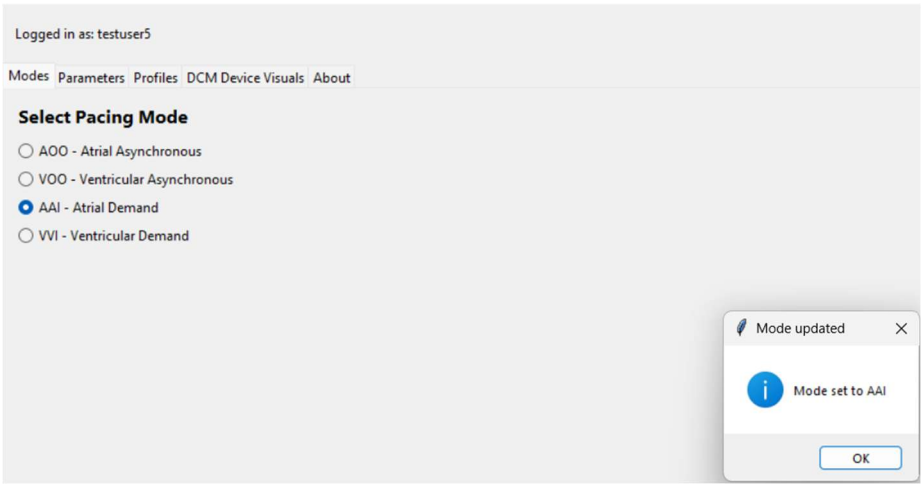


Figure 24: Mode set to AAI

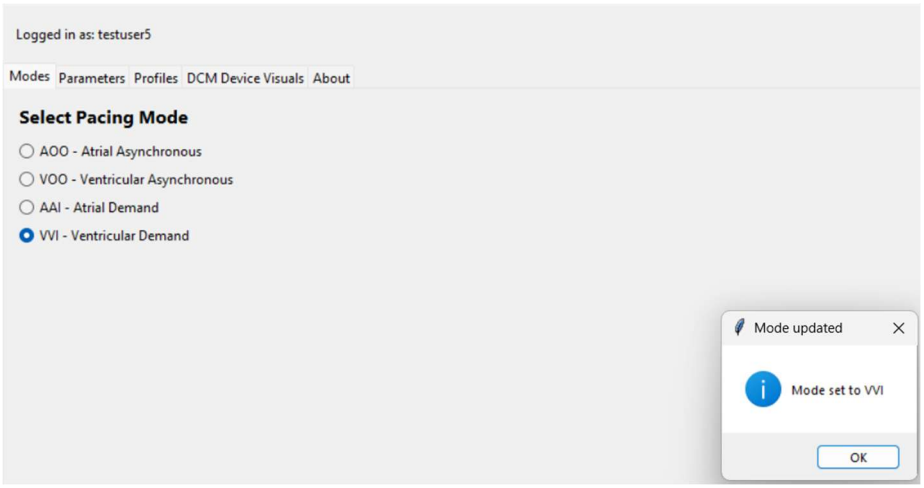


Figure 25: Mode set to VVI

5. **Result**
   Pass. Mode selection matched program logic and file updates.

Parameter Input Validation Test (parameters_view.py, app.py)

1. **Purpose**
   To validate that numeric entry fields accept only valid ranges, that type conversion occurs correctly, and that invalid inputs are rejected with detailed error messages.
2. **Input conditions**
   Valid Set and Apply Changes: LRL 60, URL 120, AtrialAmplitude 3.5, AtrialPulseWidth 1.0, VentricularAmplitude 3.5, VentricularPulseWidth 1.0, ARP 250, VRP 320.
   Invalid Set and Apply Changes: LRL 25, URL 120 (below allowed limit).
3. **Expected output**
   - Valid input displays dialog: “Saved” with text “Parameters updated and saved as last-used for this user.”
   - JSON profiles.testuser5.__last__ updated with numeric values and converted floats.
   - Invalid input displays dialog: “Invalid parameters” with text “LRL should be between 30 and 175 ppm.” and prevents file modification.
4. **Actual output**
   - For valid input, displayed:

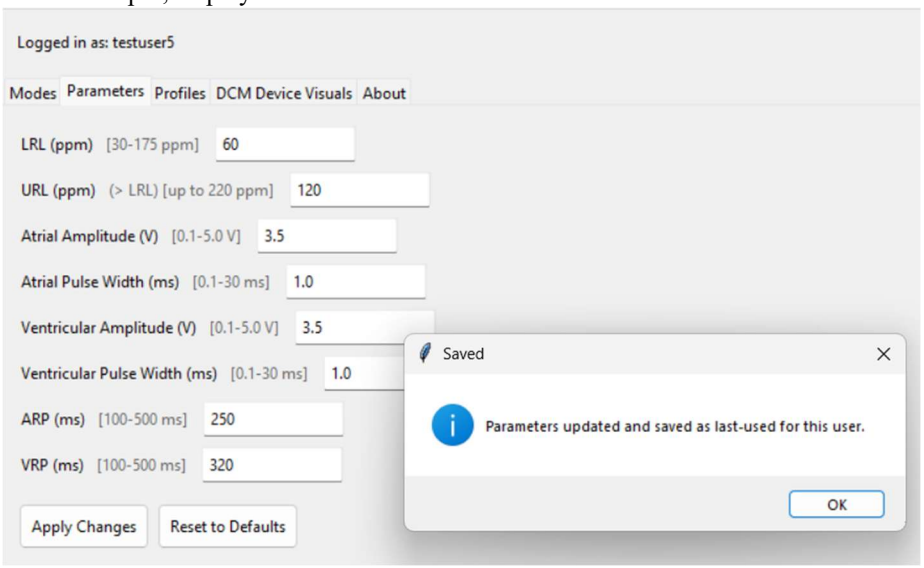


Figure 26: Parameters valid input accepted

- dcm_config.json updated:

Figure 27: dcm_config.jsoon updated after applying parameters

- Invalid case displayed:

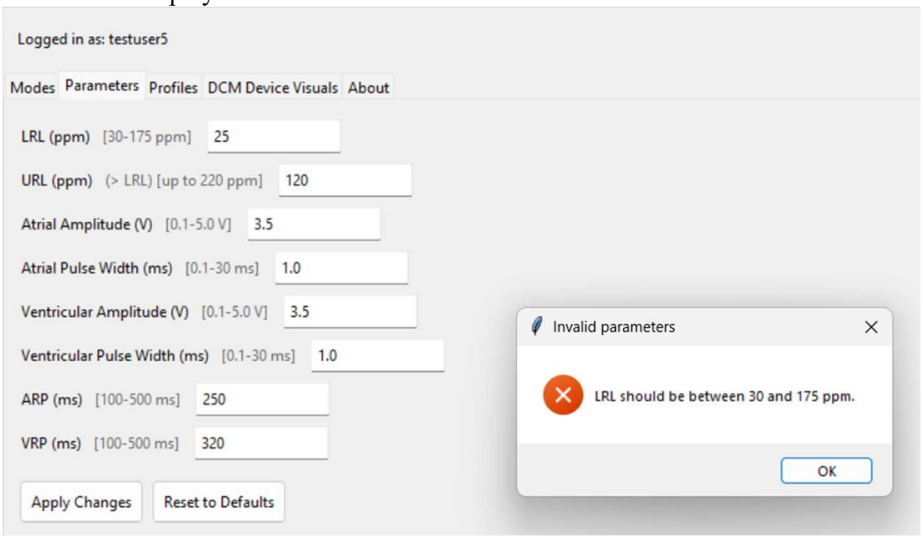


Figure 28: Parameters invalid input error message

- No file change observed.

5. Result

Pass. Validation correctly enforced range limits, data types, and persistent storage.

Profile Save Load Delete Test (profiles_view.py)

1. **Purpose**
   To ensure that users can save, load, and delete profiles, and that each operation provides correct user feedback and file modifications.
2. **Input conditions**
   Logged in as testuser5.
   Save profile name ProfileTest1.
   Load the saved profile.
   Delete the same profile.
3. **Expected output**
   - On save: dialog "Saved" with text "Profile 'ProfileTest1' saved for testuser5."
   - On load: dialog "Loaded" with text "Profile 'ProfileTest1' loaded.
   - On delete: prompt "Delete profile 'ProfileTest1'?" followed by removal from JSON.
4. **Actual output**
   - Save displayed:

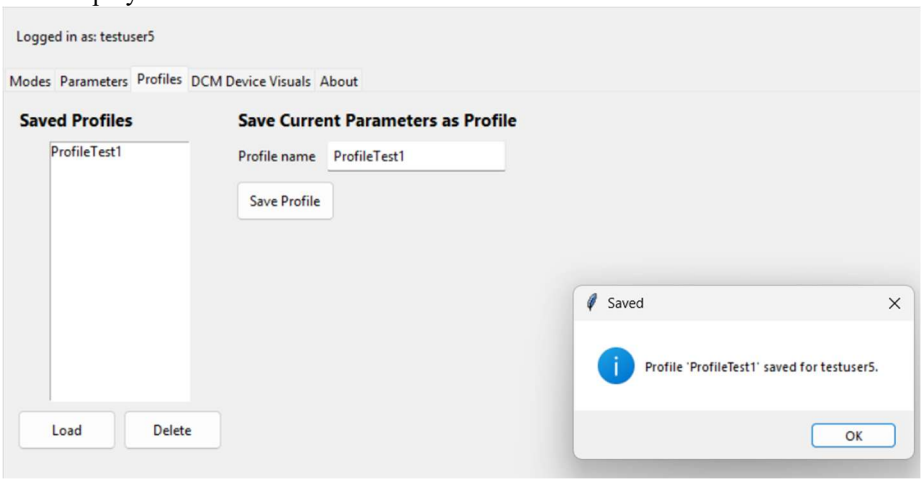


Figure 29: Profiles save confirmation dialog

- File updated:

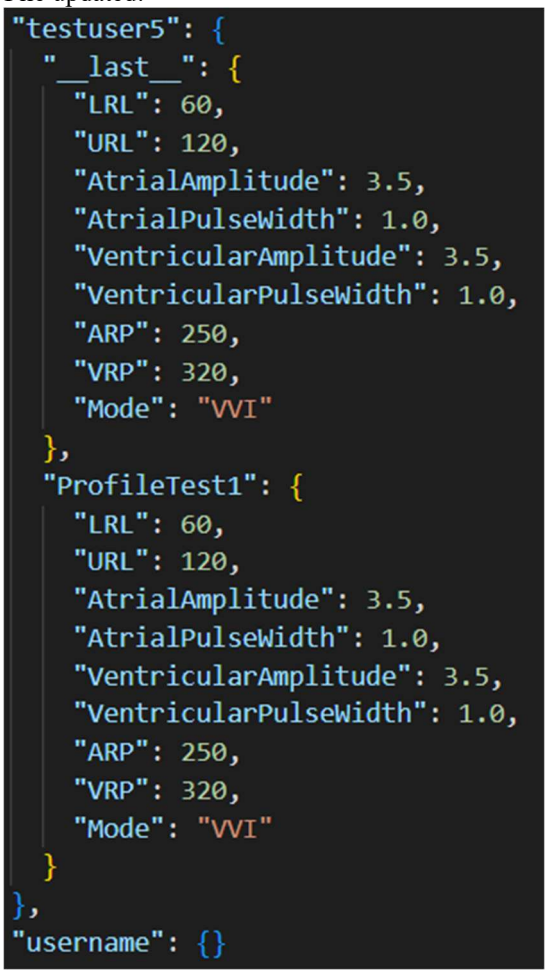


Figure 30: Profiles file updated after save

- Load displayed:

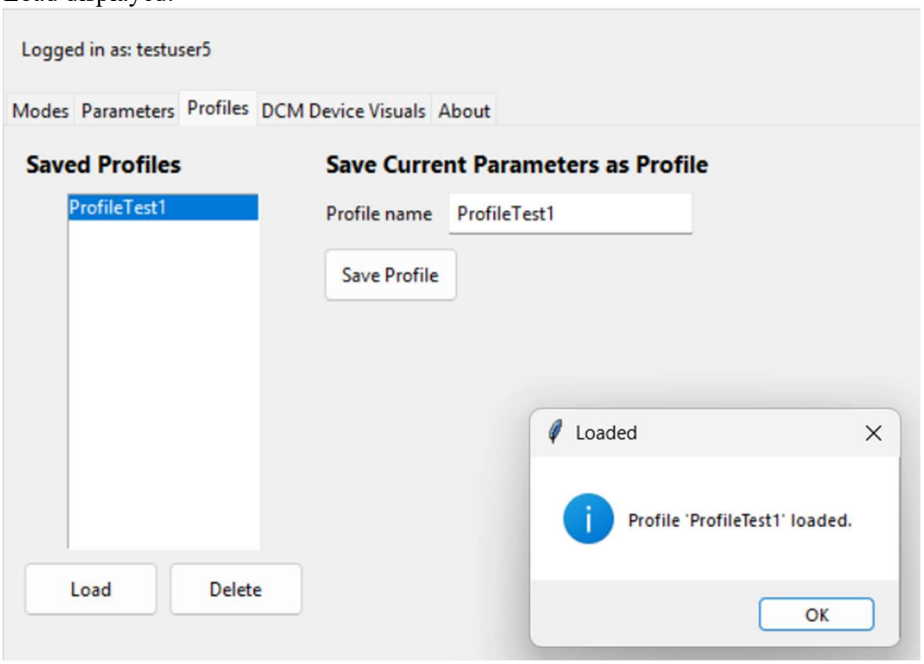


Figure 31: Profiles load dialog and available profile list

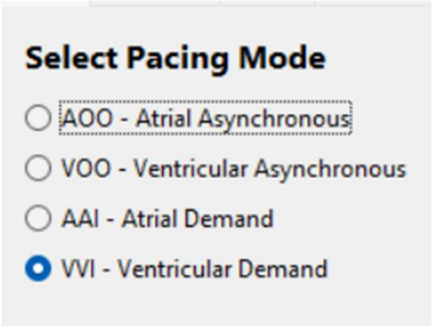


Figure 32: Profiles load pacing mode confirmation

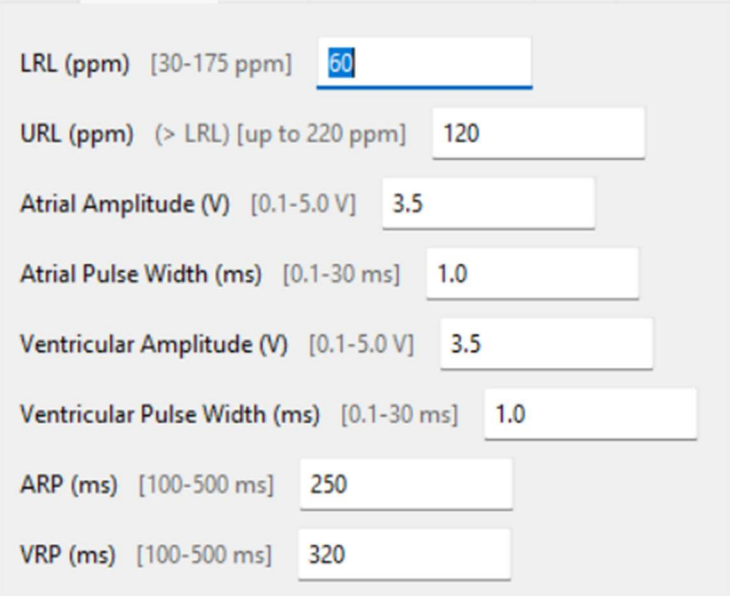


Figure 33: Profiles load parameters confirmation

- Delete displayed confirmation prompt and successfully removed entry from file:

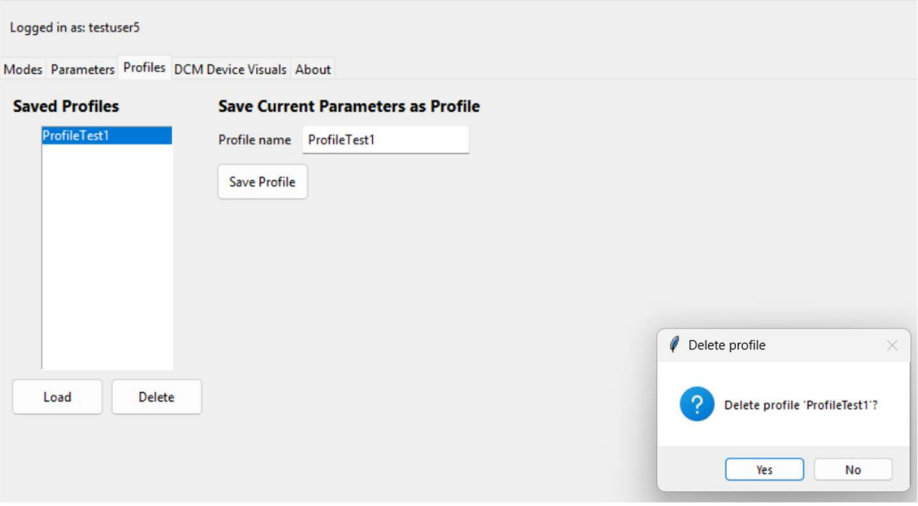


Figure 34: Profiles delete confirmation and removal from file

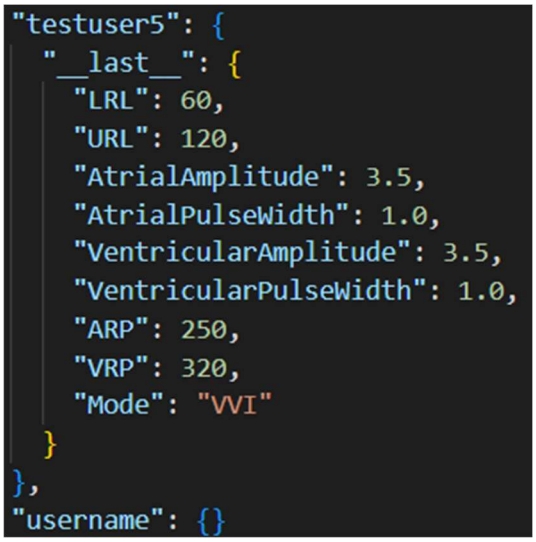


Figure 35: Profiles post-delete state of list and file

5. **Result**
   Pass. Save, load, and delete executed with correct dialogs and persistent effects.

Device Communication and Detection Test (dcm_visuals_view.py)

1. **Purpose**
   To verify that the DCM interface accurately reflects the connection and detection states through text and colour-coded indicators.
2. **Input conditions**
   Toggle sequence: Disconnect → Connect → Detect Pacemaker → Clear Pacemaker.
3. **Expected output**
   - Disconnect sets “Device Communication” text: “Device not communicating” (red).
   - Connect sets “Device Communication” text: “Device is communicating” (green).
   - Detect Pacemaker sets “Other Pacemaker Detection” text: “Different pacemaker detected in range” (orange).
   - Clear Pacemaker sets “Other Pacemaker Detection” text: “No other pacemaker in range” (gray).
4. **Actual output**

- Labels updated accordingly:

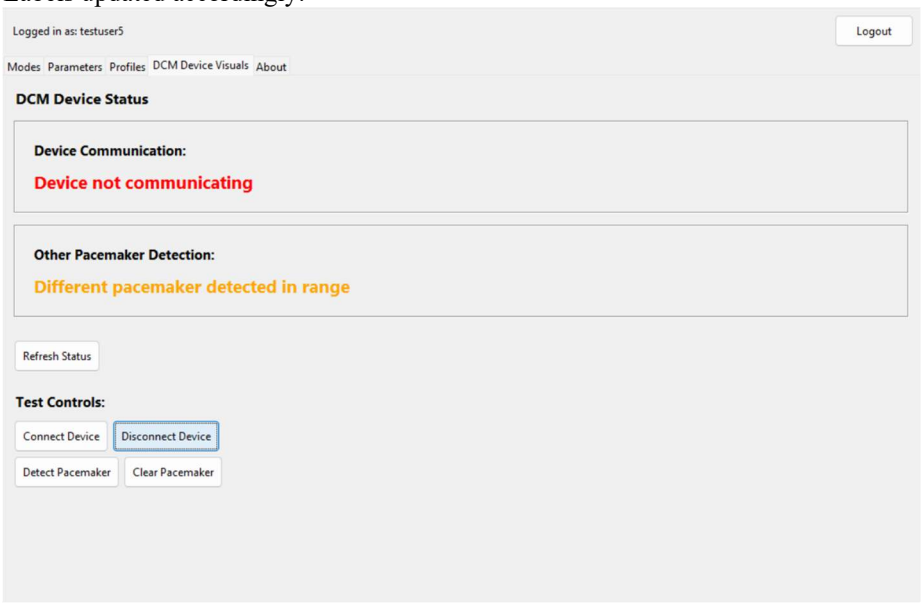


Figure 36: Disconnect device message

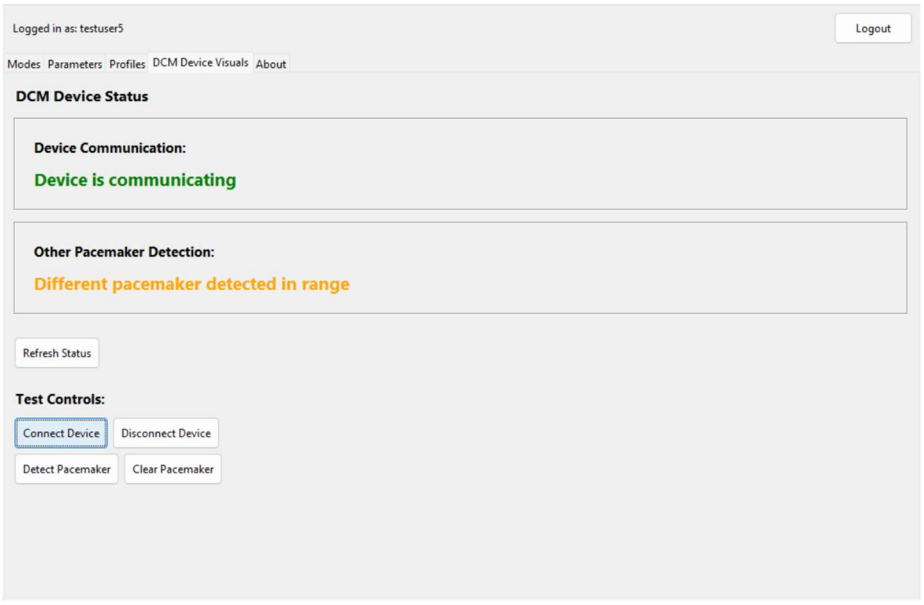


Figure 37: Connect device message

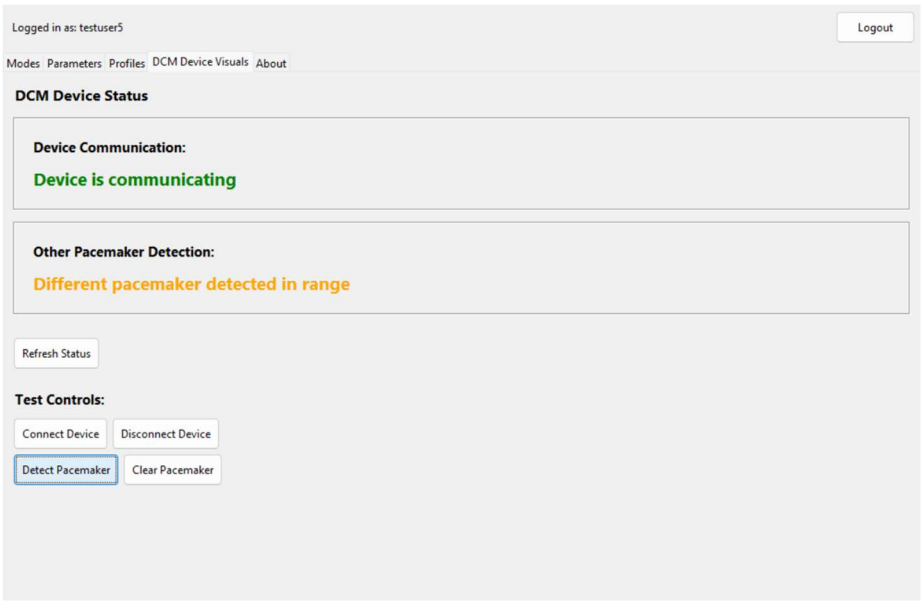


Figure 38: Detect pacemaker message

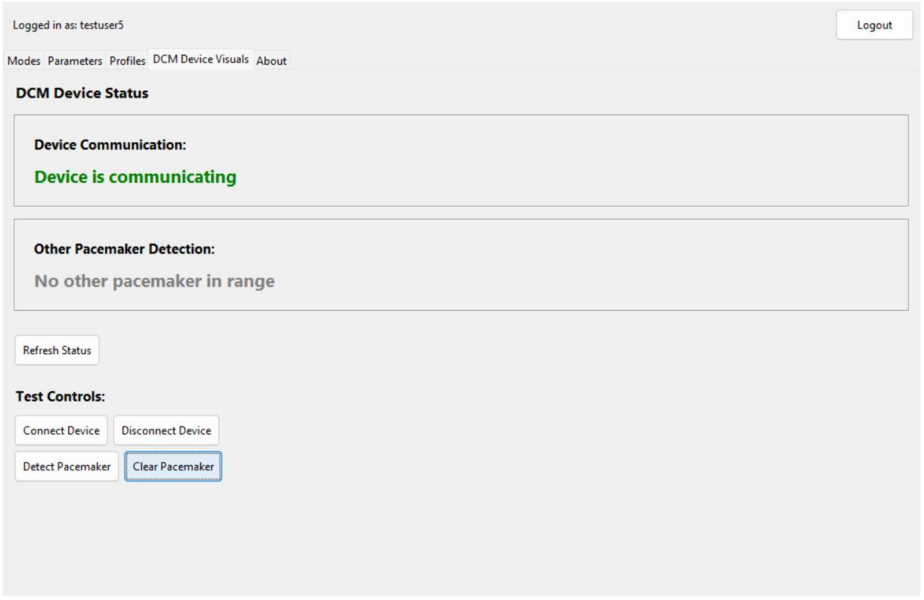


Figure 39: Clear pacemaker message

- Colours matched respective states.

5. **Result**
   Pass. Visual labels and state colours accurately reflected internal variable flags.

About Tab Verification Test (about_view.py)

1. **Purpose**
   To verify that the About tab loads properly and displays relevant application and system information.
2. **Input conditions**
   Open the About tab from the DCM dashboard.
3. **Expected output**
   - A tab that shows the text block beginning with "Device Controller-Monitor (DCM) – Deliverable 1" and listing features and storage notes.
4. **Actual output**
   - The tab displayed the exact text from about_view.py without any distortion or truncation:

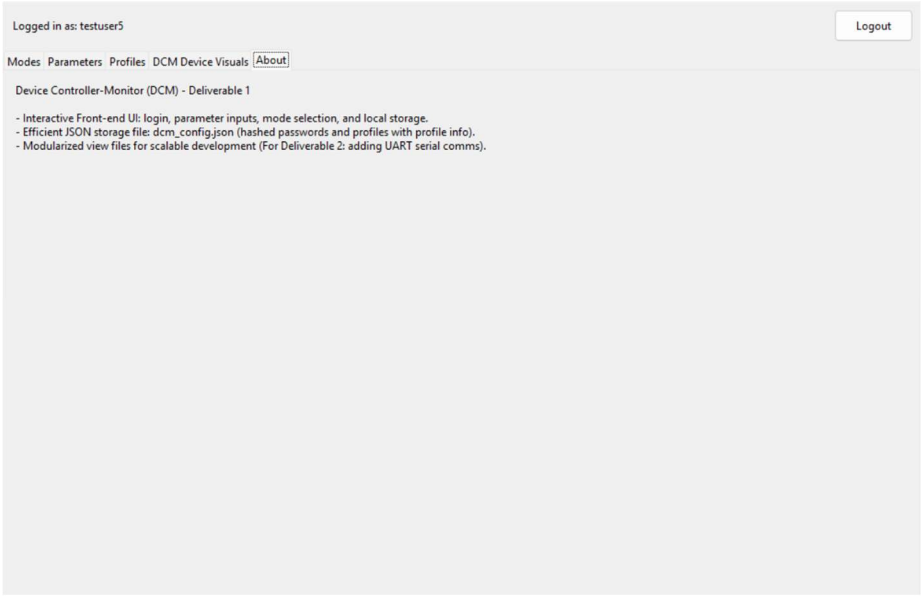


Figure 40: About tab text correctly rendered

   - Proper alignment and no layout errors.

5. **Result**
   Pass. About view displayed full text block as implemented with correct line breaks.

3.4.2 Model Tests

Atrial Sensing Threshold Test (AAI)

1. **Purpose**
   To verify that the pacemaker detects the natural atrial activity and adjusts the pacing accordingly, pacing the atrium only when the defined rate (LRL) is missed.
2. **Input conditions**
   Mode: AAI
   LRL/Pacemaker Rate: 60BPM
   Natural Atrium: ON
   Natural Ventricle: OFF
   Natural Atrial Pulse Width: initially 20ms, gradually reduced
   Natural Heart Rate: 30BPM
3. **Expected output**
   - With detectable atrial activity (20ms pulse width), the pacemaker fills the missing beats
   - As the atrial pulse width is decreased in HeartView, the natural heartbeat becomes too short/weak to detect, so the pacemaker paces at a steady 60BPM
4. **Actual output**
   The pacemaker fills missing beats from atrial pulse widths of 20ms - 9ms

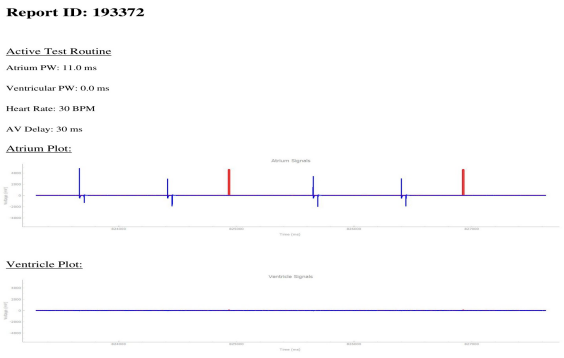


Figure 41: Normal heartbeat sensing

When the pulse width was decreased to 8ms, the pacemaker missed a detection of the heartbeat and paced without waiting the full hearbeat time:

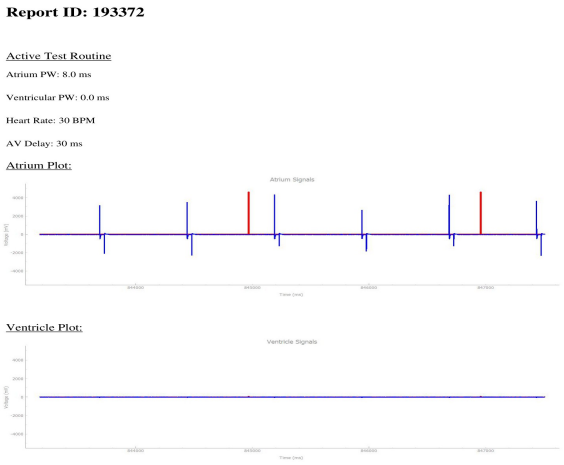


Figure 42: Not detecting heartbeats

5. **Result**
   The test was passed, as the pacemaker's pacing rate increased as the natural atrial signal became undetectable.

Ventricular Sensing Threshold Test (VVI)

1. **Purpose**
   To verify that the pacemaker detects the natural ventricular activity and adjusts the pacing accordingly, pacing the ventricular only when the defined rate (LRL) is missed.
2. **Input conditions**
   Mode: VVI
   LRL/Pacemaker Rate: 60BPM
   Natural Atrium: OFF
   Natural Ventricle: ON
   Natural Ventricular Pulse Width: initially 20ms, gradually reduced
   Natural Heart Rate: 30BPM
3. **Expected output**
   - With detectable ventricular activity (20ms pulse width), the pacemaker fills the missing beats

- As the ventricular pulse width is decreased in HeartView, the natural heartbeat becomes too short/weak to detect, so the pacemaker paces at a steady 60BPM

4. Actual output

The pacemaker fills missing beats from atrial pulse widths of 20ms - 9ms

Report ID: 193372

Active Test Routine

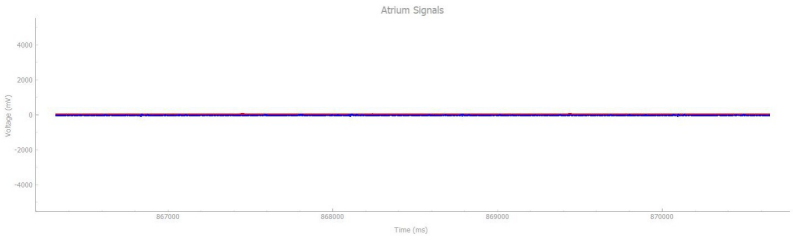
Atrium PW: 0.0 ms

Ventricular PW: 19.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Ventricle Plot:

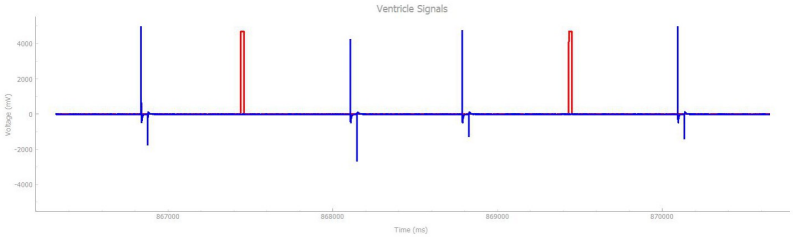


Figure 43: Normal heartbeat sensing

When the pulse width was decreased to 8ms, the pacemaker missed a detection of the heartbeat and paced without waiting the full hearbeat time:

Report ID: 193372

Active Test Routine

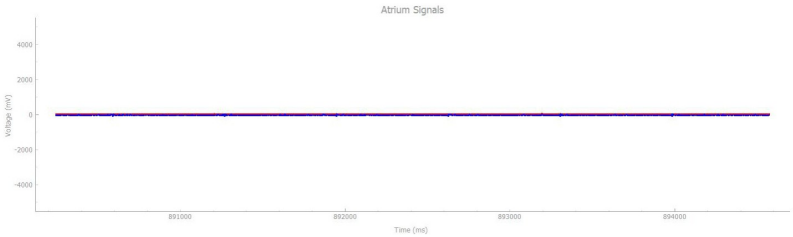
Atrium PW: 0.0 ms

Ventricular PW: 8.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Ventricle Plot:

Figure 44: Not detecting heartbeats

5. **Result**

The test was passed, as the pacemaker's pacing rate increased as the natural ventricular signal became undetectable.

Asynchronous Atrial Pacing Test (AOO)

1. **Purpose**
   To verify that the pacemaker atrial pacing is asynchronous, with the device pacing the atrium at the programmed rate and pulse width regardless of the natural heartbeat
2. **Input conditions**

Mode: AOO
LRL/Pacemaker Rate: 60BPM
Natural Atrium: ON
Natural Ventricle: OFF
Natural Heart Rate: 30BPM

3. **Expected output**
   - Blue atrial pacing spikes at a constant 60BPM (~1000ms period), independent of the natural heart beat (red spikes)
4. **Actual output**

Matches expected output and artificially paces at a constant 60BPM

Report ID: 193372

Active Test Routine

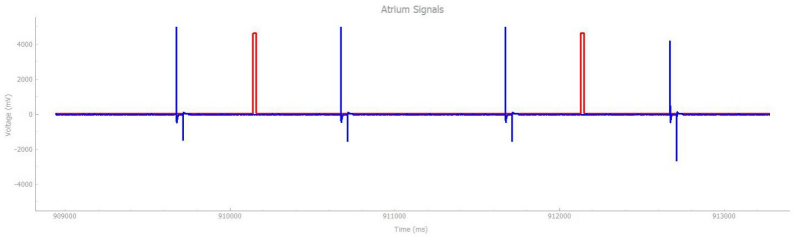
Atrium PW: 20.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Ventricle Plot:

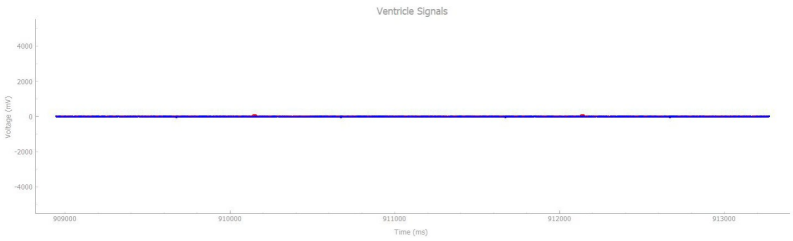


Figure 45: Paces atrium every ~1000ms

5. **Result**

The test was passed, with the actual output matching the expected and pacing the atrium properly

Asynchronous Ventricular Pacing Test (VOO)

1. **Purpose**
   To verify that the pacemaker ventricular pacing is asynchronous, with the device pacing the ventricular at the programmed rate and pulse width regardless of the natural heartbeat
2. **Input conditions**
   Mode: AOO
   LRL/Pacemaker Rate: 60BPM
   Natural Atrium: OFF
   Natural Ventricle: ON
   Natural Heart Rate: 30BPM
3. **Expected output**
   - Blue ventricular pacing spikes at a constant 60BPM (~1000ms period), independent of the natural heart beat (red spikes)
4. **Actual output**
   Matches expected output and artificially paces at a constant 60BPM

Report ID: 193372

Active Test Routine

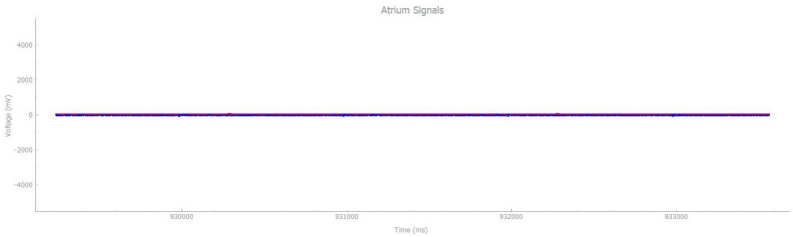
Atrium PW: 0.0 ms

Ventricular PW: 20.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Atrium Plot:



Ventricle Plot:

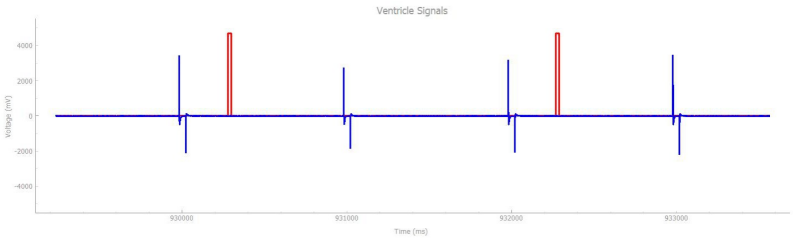


Figure 46: Paces atrium ~1000ms (continued)

5. **Result**
   The test was passed, with the actual output matching the expected and pacing the ventricular properly

3.5 GenAI Usage

GenAI tools were not used at any stage of this deliverable (project). All development, testing, and documentation were completed manually by the team. The code, interface design, and written content were produced entirely through the team's own effort and collaboration.