SFWRENG 3K04

Deliverable 1

Group 28

Fall 2025

Contents

1 Group Members

Aaditya Anil, anila4, 400520758

Deev Patel, pated201

Shaun Plassery, plasers,

Dylan Manamendra, manamenk, 400509187

2 Part 1

2.1 Introduction

Arrythmia is a conduction disorder in which the heart's electrical signal is disrupted inducing, irregular heart rhythm or rate. Individuals with a form of arrythmia are at great risk of heart attacks, cardiac arrest, organ failure, and stroke (NHLBI, NIH – March 2022). To counteract this, a pacemaker is used to detect when the heart rate of the user drops, then administer a controlled electrical signal to the muscles surrounding the heart to restabilize its natural rhythm. The pacemaker being designed currently is designed to treat a form of arrythmia known as Bradycardia, in which the heart rhythm drops to a dangerously low rate (<60 bpm). The pacemaker is equipped with a suite of sensors to detect information on the user's heart rate, checking if said info is in line with the conditions of those experiencing an arrhythmic episode. Once an arrhythmic episode is detected, the pacemaker administers a controlled electrical signal to the surrounding muscular tissue, forcing the heart to beat at an appropriate rate.

The Device Control Module (DCM) is the software interface that enables clinicians to configure and monitor the pacemaker device. It provides user authentication, pacing-mode configuration, programmable parameter entry, and placeholder functionality for egram visualization. For this deliverable, only the presentation layer (front-end) of the DCM has been implemented; communications between the DCM and pacemaker hardware are planned to be incorporated in the next phase

This first deliverable is to document the initial requirements of the project, and ensure that the group members not only understand the function of the pacemaker, but are aware of the necessary specifications needed to design and construct the state flow diagrams of the pacemaker and the graphical user interface (GUI) for the DCM.

2.2 Requirements

- Pulse width and refractory period timing of atrium and ventricle
- Information on the low-rate limit and upper-rate limit (LRL, URL)
- Hysteresis (irregularity in pulse rate) state detection
- Pacing capacitor with sufficient power source

o AOO: grounding control (Pace Cap/Ventricle/Atrium), pace control (Atrium, Ventricle), DCM control, LRL, URL, atrial pulse width

o VOO: grounding control, pace control (Atrium, Ventricle), DCM control, LRL, URL, ventricle pulse width

o AAI: grounding control, pace control (Atrium, Ventricle), DCM control, LRL, URL, hysteresis state, atrium pulse width, atrial refractory period

- VVIL grounding control, pace control (Atrium, Ventricle), DCM control, LRL, ventricle pulse width, hysteresis state, ventricle refractory period

The following summarizes how the implemented DCM fulfills **section 3.2.2 of the PACEMAKER System Specification**:

Req ID	Description	Type	Trace to Design Section
R1	The DCM shall allow registration and login for up to 10 users.	Functional	DCM Design § User Utilities
R2	The DCM shall display and edit programmable parameters (LRL, URL, AA, APW, VA, VPW, VRP, ARP).	Functional	DCM Design § Parameter Manager
R3	The DCM shall provide a status indicator for device connection.	Functional	DCM Design § DeviceManager
R4	The DCM shall save all parameter data locally in JSON format.	Functional	DCM Design § File Persistence
R5	The DCM shall indicate when a different device is detected.	Functional	DCM Design § DeviceManager
R6	The system shall validate all numeric parameters against range limits.	Non-Functional (Safety)	DCM Design § Parameter Validation

Additional derived requirements from the assignment:

- Store ≤ 10 local users.
- Provide input validation for parameter ranges (Section `PARAM_RANGES`).
- Allow saving and loading of parameter files.
- Maintain a placeholder for future Egram recording and plotting.

2.3 Design

The state flow model, designed using Simulink, illustrates the direction and behavior of data within the pacemaker device, highlighting which states use specific information (LRL, atrium pulse width, etc.) for their operation.

The pacemaker is fed information regarding the LRL and URL, the mode of operation, as well as sensor data detecting the atrial and ventricle compression and pulse width, atrial and ventricle refractory periods, and whether the heart rate has entered hysteresis.

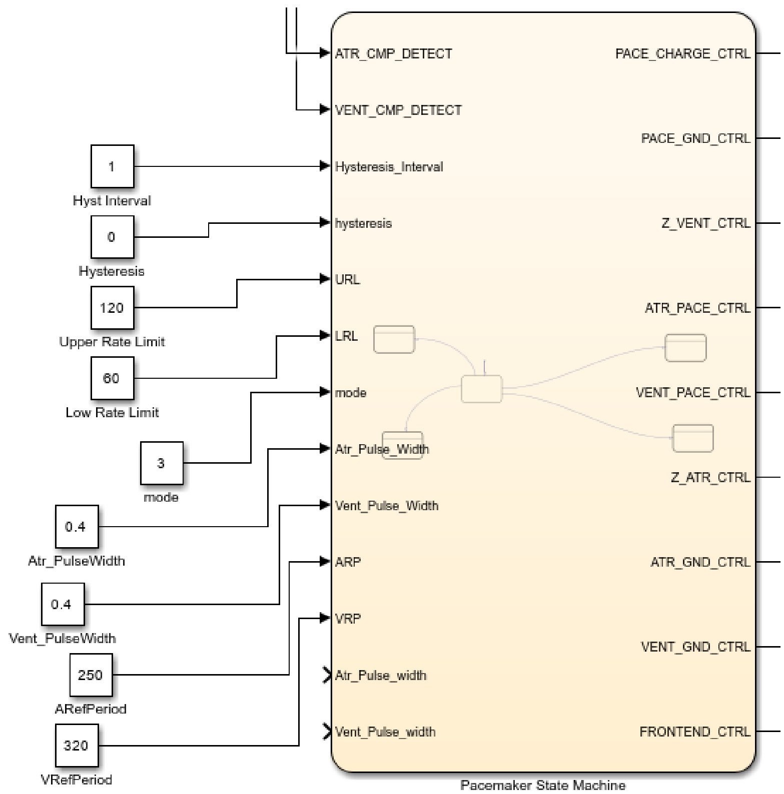

Figure 1: Pacemaker State Machine with external input data

The mode variable is responsible for setting the state of the pacemaker, and subsequently the means in which it functions.

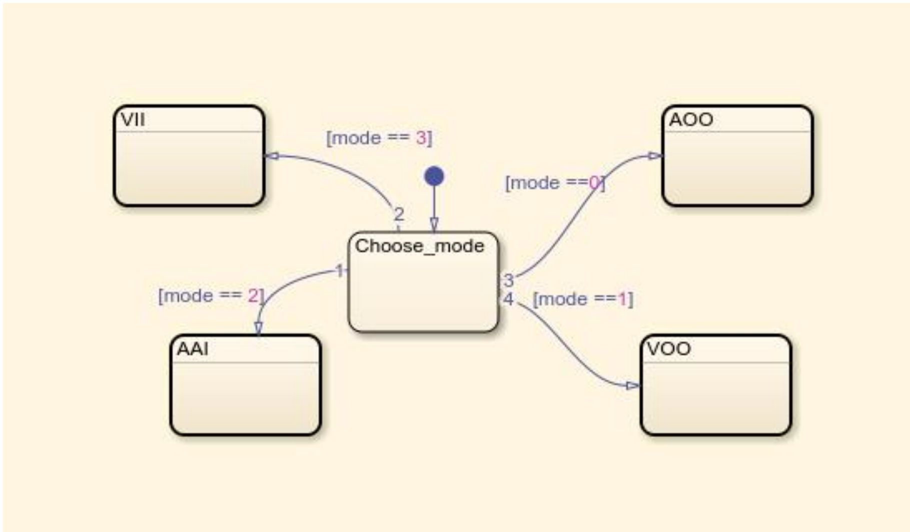

Figure 2: Mode value used as state control for pacemaker

Within each state is the parameters and data used for the specific function, as well as transition conditions between specific operations within the state. When entering the AOO state, the state flow starts a timer; upon reaching the computed LRL interval it declares atrial pace control, ATR_PACE_CTRL, for Atr_PulseWidth, then resets the interval. Additionally, it allows for the pacing capacitor to be charged up. No sensing is performed (inhibit = none), and the pacemaker enters “no response” behavior. AOO applies only to the atrial activity.

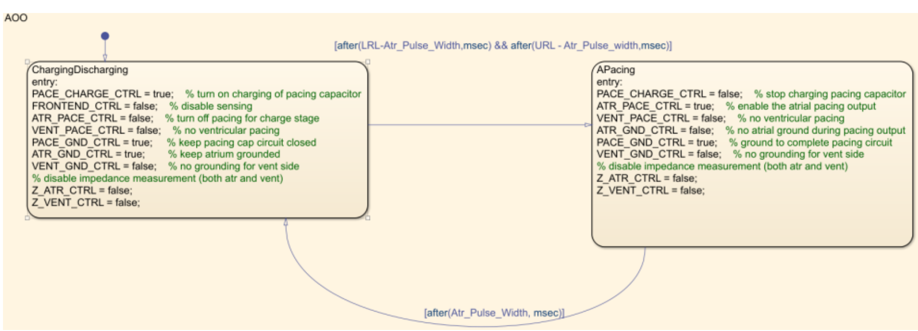

Figure 3: AOO state of pacemaker

When entering the VOO state, the state flow once again starts a timer; upon reaching the computed LRL interval it declares the ventricle pace control, VENT_PACE_CTRL, for Vent_PulseWidth, then resets the interval whilst also charging the pacing capacitor. The function of VOO is similar to AOO, except applying to the ventricle activity as opposed to the atrial activity.

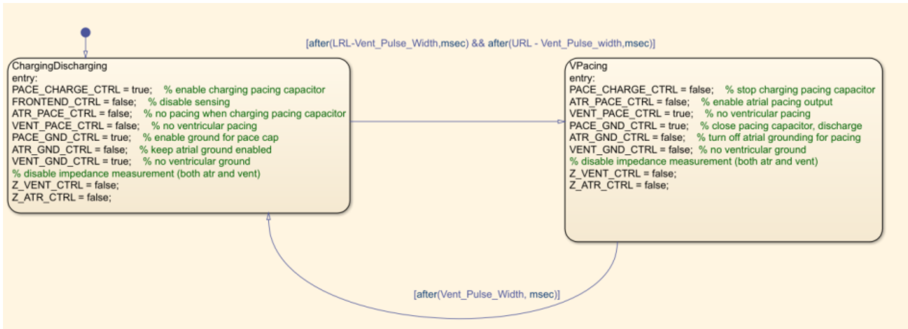

Figure 4: VOO state of pacemaker

When entering the AAI state, the controller starts the LRL timer. If ATR_CMP_DETECT (Boolean to detect atrial compression) occurs before the timer elapses and not in refractory, the timer is reset, and no pace is delivered. If the timer completes without an atrial sense, the state flow declares ATR_PACE_CTRL for Atr_PulseWidth. The pacemaker will also discharge the pacing capacitor, and allow it to charge up again. This once again, is only with regards to atrial activity

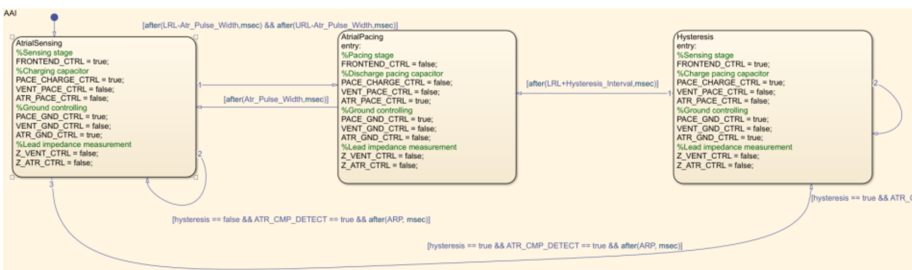

Figure 5: AAI state for pacemaker

When entering the VVI state, the pacemaker computes f_waitInterval based on p_lowrateInterval (found using LRL) plus hysteresis if enabled, account for ventricular refractory periods (VRP), and detect spontaneous ventricular sense in the wait interval. If detected, a ventricular pace is delivered (VENT_PACE_CTRL) by pacing capacitors, so long as neither sensed nor paced events occur within the wait interval. The chart implements In_vPace, In_pVRP, and the c_vp output logic as described in the pacemaker document.

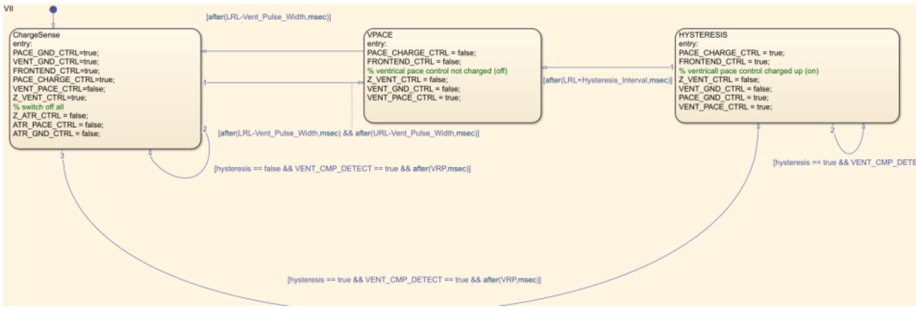

Figure 6: VVI state of the pacemaker

In all states of the pacemaker, conditions such as after() and values such as LRL, URL, vent/atr pulsewidth, and internal timers are used to implement intervals and pulse widths (e.g., after(LRL - Vent_Pulse_Width, msec) for pacing decision timing and after(p_vPaceWidth, msec) to clear a pacing pulse). These match the timings and tolerances for pulse width and refractory intervals labeled on the pacemaker documentation. Additionally, expectations such as asynchronous operation of AOO and VOO states, AAI and VVI control, manual control via the DCM, and Hardware hiding to correct board ports (for simulation of an actual heart) have been met in accordance to the deliverable 1 expectations.

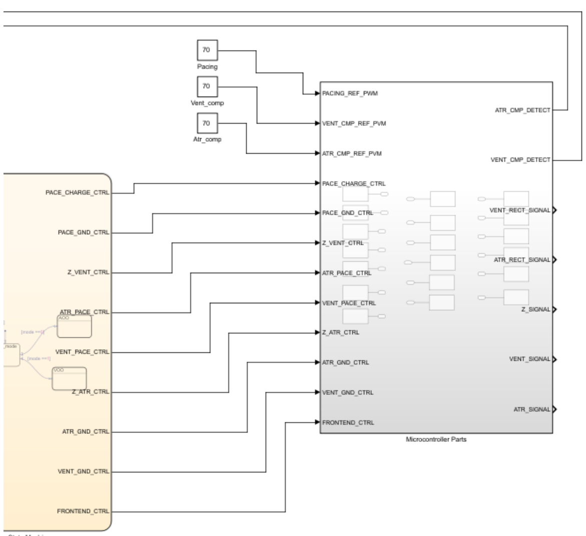

Figure 7:Hardware hiding of outputs to correct ports in microcontroller

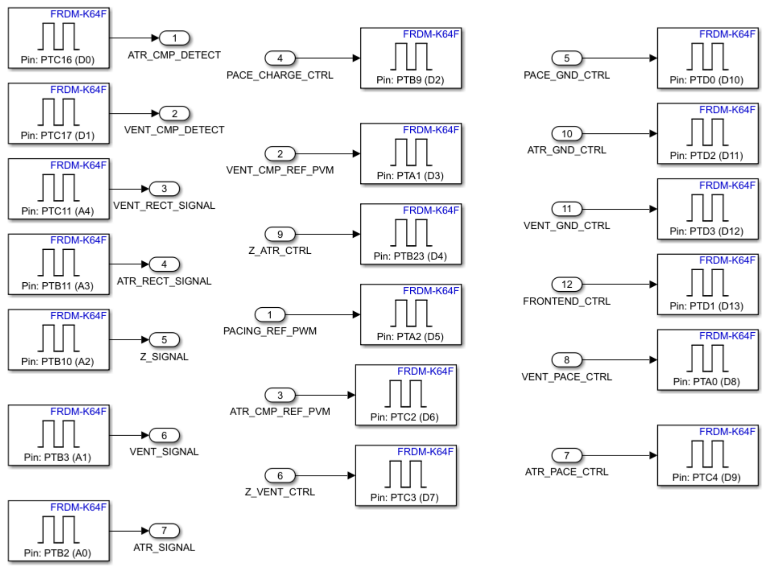

Figure 8: Port connection of specific waveforms within the microcontroller

The DCM follows a **modular, object-oriented structure**, divided into self-contained components for user management, device management, parameter storage, and data visualization.

Major Modules

- **WelcomeApp (class)** – Main Tkinter application controller; manages frame navigation and state.
- **DeviceManager (class)** – Tracks the most recently interrogated device to support requirement (7).
- **User Management Utilities** – Handle registration and authentication with password hashing and salt.
- **Parameter Management Subsystem** – Validates and stores programmable parameters per mode.
- **Egram Subsystem** – Defines JSON schema and save/list/preview functions for recorded Egrams.

Below the User Login → Main DCM → Parameter Manager → Egram Module block diagram can be found.

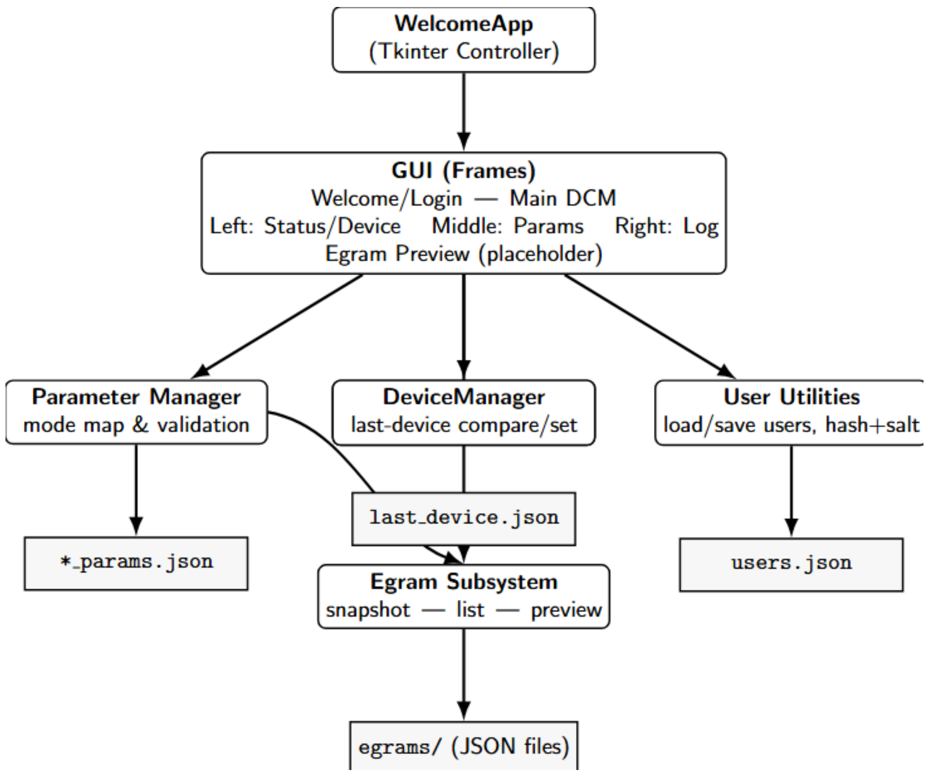

Figure 9: Block Diagram of DCM

The welcome screen offers three options — Login, Register, or Exit. Registration supports up to 10 users stored locally in `users.json`. Passwords are hashed with SHA-256 and per-user salts for security.

```python
class WelcomeApp(tk.Tk):
    # ----------------------
    # Welcome / login / register
    # ----------------------
    def _build_welcome(self):
        f = self.frame_welcome
        tk.Label(f, text="Welcome!", font=("Arial", 18)).pack(pady=(0, 10))
        tk.Button(f, text="Login", width=20, command=lambda: self.show_frame(self.frame_login)).pack(pady=5)
        tk.Button(f, text="Register", width=20, command=lambda: self.show_frame(self.frame_register)).pack(pady=5)
        tk.Button(f, text="Exit", width=20, command=self.quit).pack(pady=(10, 0))

    def _build_login(self):
        f = self.frame_login
        tk.Label(f, text="Login", font=("Arial", 14)).grid(row=0, column=0, columnspan=2, pady=(0, 8))
        tk.Label(f, text="Username:").grid(row=1, column=0, sticky="e")
        self.login_user = tk.Entry(f); self.login_user.grid(row=1, column=1)
        tk.Label(f, text="Password:").grid(row=2, column=0, sticky="e")
        self.login_pass = tk.Entry(f, show="*"); self.login_pass.grid(row=2, column=1)
        self.login_msg = tk.Label(f, text="", fg="red"); self.login_msg.grid(row=3, column=0, columnspan=2)
        tk.Button(f, text="Submit", width=12, command=self.attempt_login).grid(row=4, column=0, pady=8)
        tk.Button(f, text="Back", width=12, command=lambda: self.show_frame(self.frame_welcome)).grid(row=4, column=1)

    def _build_register(self):
        f = self.frame_register
        tk.Label(f, text="Register", font=("Arial", 14)).grid(row=0, column=0, columnspan=2, pady=(0, 8))
        tk.Label(f, text="Username:").grid(row=1, column=0, sticky="e")
        self.reg_user = tk.Entry(f); self.reg_user.grid(row=1, column=1)
        tk.Label(f, text="Password:").grid(row=2, column=0, sticky="e")
        self.reg_pass = tk.Entry(f, show="*"); self.reg_pass.grid(row=2, column=1)
        tk.Label(f, text="Confirm:").grid(row=3, column=0, sticky="e")
        self.reg_pass2 = tk.Entry(f, show="*"); self.reg_pass2.grid(row=3, column=1)
        self.reg_msg = tk.Label(f, text="", fg="red"); self.reg_msg.grid(row=4, column=0, columnspan=2)
        tk.Button(f, text="Create", width=12, command=self.attempt_register).grid(row=5, column=0, pady=8)
        tk.Button(f, text="Back", width=12, command=lambda: self.show_frame(self.frame_welcome)).grid(row=5, column=1)

    def attempt_register(self):
```

Figure 10: Code for login window

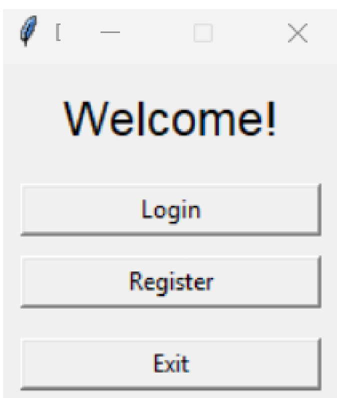

Figure 11: Login window

Upon login, the user is brought to the main DCM interface, which contains three panes:

- **Left Pane:** Connection status indicator, device dropdown, and refresh/disconnect buttons.

- **Middle Pane:** Dynamic parameter fields for all programmable parameters.
- **Right Pane:** Communication log panel displaying DCM events.

The top bar shows the logged-in user and a logout button.

```
class WelcomeApp(tk.Tk):
    def _build_dcm(self):

        # Right: packet log
        right = tk.Frame(f)
        right.pack(side="left", padx=(12,0))
        tk.Label(right, text="Comm Log", font=("Arial", 10)).pack(pady=(0,4))
        self.log_text = tk.Text(right, width=48, height=16, state="disabled")
        self.log_text.pack(pady=(4,0))

        # placeholder for egram viewer + quick egram actions
        frame_eg = tk.Frame(f)
        frame_eg.pack(side="bottom", pady=(12,0))
        tk.Label(frame_eg, text="Egram Viewer (placeholder)", fg="gray").pack(side="left")
        # quick egram snapshot / list buttons
        try:
            tk.Button(frame_eg, text="Save Egram Snapshot", command=self._ui_save_egram).pack(side="left", padx=(8,4))
        except Exception:
            # if ttk isn't imported, fall back
            btn = tk.Button(frame_eg, text="Save Egram Snapshot", command=self._ui_save_egram)
            btn.pack(side="left", padx=(8,4))
        try:
            tk.Button(frame_eg, text="List Egrams", command=self._ui_list_egms).pack(side="left", padx=(4,0))
        except Exception:
            btn2 = tk.Button(frame_eg, text="List Egrams", command=self._ui_list_egms)
            btn2.pack(side="left", padx=(4,0))

        # initial device list
        self._refresh_device_list()
        # trace selection
        self.device_var.trace_add("write", self._on_device_selected)
```

Figure 12: Code for Main DCM Interface

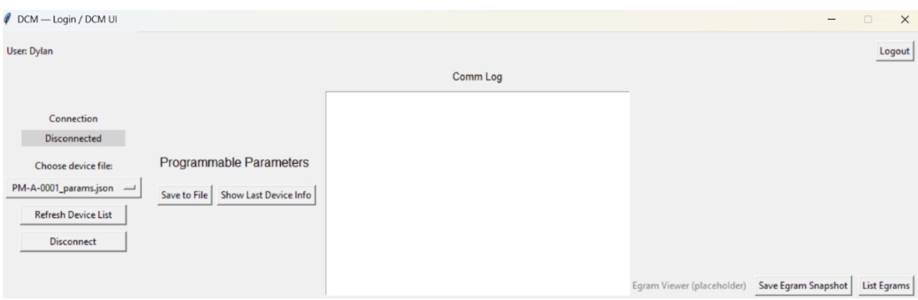

Figure 13: Main Page for DCM

Each mode (e.g., AOO, VVI, DDD) enables only its relevant parameters based on the mapping in `PARAMS_BY_MODE`.
Entries are validated through `_clamp_to_range()` and `_is_number()` before saving.
If a parameter is out of range (e.g., LRL > URL or amplitude > 7.5 V), an error dialog appears via `messagebox.showerror()`.

```
class WelcomeApp(tk.Tk):
    def _build_dcm(self):
        # Middle: parameter viewer / editor
        mid = tk.Frame(f)
        mid.pack(side="left")
        tk.Label(mid, text="Programmable Parameters", font=("Arial", 12)).pack()

        self.params_frame = tk.Frame(mid, bd=1, relief="sunken", padx=6, pady=6)
        self.params_frame.pack(pady=(6,4))

        # Parameter entries will be created dynamically
        self.param_vars = {}
        self.param_entries = {}

        btns = tk.Frame(mid); btns.pack(pady=6)
        tk.Button(btns, text="Save to File", command=self._save_params_to_file).pack(side="left", padx=6)
        tk.Button(btns, text="Show Last Device Info", command=self._show_last_device_info).pack(side="left")

        # Right: packet log
        right = tk.Frame(f)
        right.pack(side="left", padx=(12,0))
        tk.Label(right, text="Comm Log", font=("Arial", 10)).pack(pady=(0,4))
        self.log_text = tk.Text(right, width=48, height=16, state="disabled")
        self.log_text.pack(pady=(4,0))

        # placeholder for egram viewer + quick egram actions
        frame_eg = tk.Frame(f)
        frame_eg.pack(side="bottom", pady=(12,0))
        tk.Label(frame_eg, text="Egram Viewer (placeholder)", fg="gray").pack(side="left")
        # quick egram snapshot / list buttons
```

Figure 14: Code for Parameter Editor

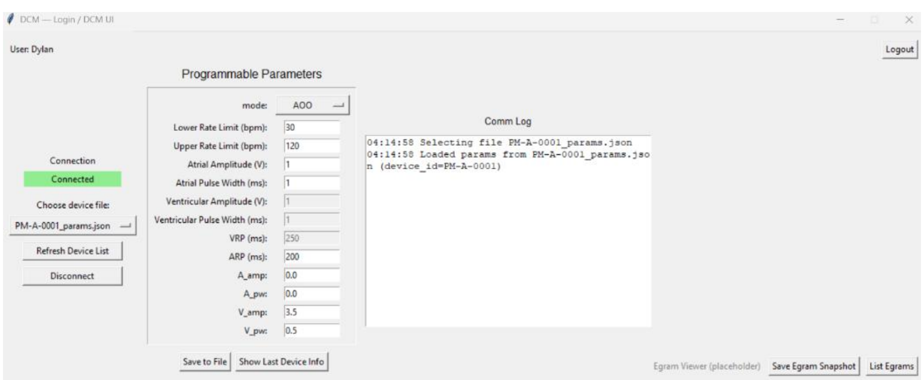

Figure 15: Parameter Editor

A status label in the left pane changes color to represent connection state:

- Gray = Disconnected
- Green = Connected
  If a new device is selected, the program compares the device ID and prompts the user to confirm switching.

```python
class WelcomeApp(tk.Tk):
    def _build_dcm(self):

        tk.Label(left, text="Connection").pack()
        self.status_label = tk.Label(left, text="Disconnected", bg="lightgray", width=18)
        self.status_label.pack(pady=(4,8))

        # Device selector dropdown
        tk.Label(left, text="Choose device file:").pack(pady=(6,2))
        self.device_var = tk.StringVar(value="(none)")
        self.device_menu = tk.OptionMenu(left, self.device_var, "(none)")
        self.device_menu.config(width=22)
        self.device_menu.pack()
        tk.Button(left, text="Refresh Device List", width=18, command=self._refresh_device_list).pack(pady=(6,2))
        # Disconnect placed under the device selector as requested
        tk.Button(left, text="Disconnect", width=18, command=self._dcm_disconnect).pack(pady=(6,2))

        # Middle: parameter viewer / editor
        mid = tk.Frame(f)
        mid.pack(side="left")
```

Figure 16: Code for status label

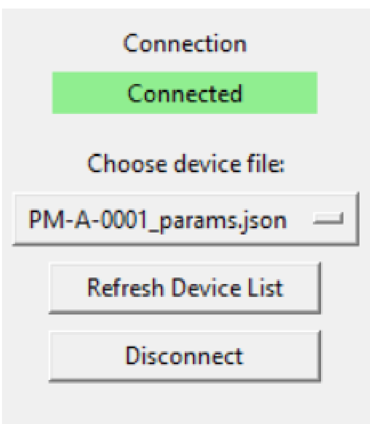

Figure 17: Status Label

The current version implements the Egram data structure and simulated waveform generation. Recorded data are saved as JSON files and previewed using Tkinter Canvas. Future iterations will support live signal streaming and real-time plotting.

```python
class WelcomeApp(tk.Tk):
    def _build_dcm(self):

        # placeholder for egram viewer + quick egram actions
        frame_eg = tk.Frame(f)
        frame_eg.pack(side="bottom", pady=(12,0))
        tk.Label(frame_eg, text="Egram Viewer (placeholder)", fg="gray").pack(side="left")
        # quick egram snapshot / list buttons
        try:
            tk.Button(frame_eg, text="Save Egram Snapshot", command=self._ui_save_egram).pack(side="left", padx=(8,4))
        except Exception:
            # if ttk isn't imported, fall back
            btn = tk.Button(frame_eg, text="Save Egram Snapshot", command=self._ui_save_egram)
            btn.pack(side="left", padx=(8,4))
        try:
            tk.Button(frame_eg, text="List Egrams", command=self._ui_list_egms).pack(side="left", padx=(4,0))
        except Exception:
            btn2 = tk.Button(frame_eg, text="List Egrams", command=self._ui_list_egms)
            btn2.pack(side="left", padx=(4,0))
```

Figure 18: Current Code for Egram Viewer

Figure 19: Current Egram Viewer

3 Part 2

3.1 Requirements Potential Changes

Scope reminder (D1): Presentation layer only (Tkinter GUI), with local user auth, local device param files, parameter validation by mode, “different device” detection, and offline egram placeholder.

Likely changes for Deliverable 2 (per assignment brief):

- **R-01 Modes & Parameters (expand):** Extend from minimal modes/params to **all required modes** and the **complete parameter set** (PACEMAKER Table 7; srsVVI §3.1 minimal set).
- **R-02 Communication (new):** Add **serial communication** to transmit/receive parameters and egram data (configure port, baud, timeouts).
- **R-03 Parameter Verification (new):** After sending parameters, **read back** from the pacemaker and verify exact match (include data typing and unit conversions; show mismatch).
- **R-04 Parameter Constraints (update):** Adopt **new ranges and increments**:
  – A/V Amplitude: Off, **0.1–5.0 V** @ **0.1 V** step (Nominal 5 V ±12%)
  – A/V Pulse Width: **1–30 ms** @ **1 ms** step
  – A/V Sensitivity: **0–5 V** @ **0.1 V** step (±2%)
- **R-05 Egram Streaming (upgrade):** Replace simulated snapshots with **live egram display** received over serial (A, V, or both channels). Allow pause/resume and snapshots.
- **R-06 Telemetry states (refine):** Extend current "Connected/Disconnected" to **timeout/error/noise/out-of-range** conditions if device exposes these.
- **R-07 Persistence (extend):** In addition to local JSON, maintain **TX/RX verification logs** (timestamped records of sent vs. read-back values).
- **R-08 Safety/UX (add):** Non-blocking UI during serial I/O, clear operator prompts on parameter step violations and verification failures.

3.2 Design Decision Potential Changes

- **DD-01 Parameter Metadata Source:**
  Current: Optional `param_metadata.json` overrides in `_load_param_metadata()`.
  Change: **Adopt metadata as authoritative** for all ranges, units, and **step sizes**, keeping code defaults as fallback only. Add field for `step` and display units in labels.
- **DD-02 Validation Model:**
  Current: Range checks via `_clamp_to_range`, numeric checks with `_is_number`, and **mode-gated** enable/disable via `_mode_allowed_params`.
  Change: Add **step-size enforcement** (e.g., multiples of 0.1 V / 1 ms). Validate **LRL ≤ URL** (already present) and any **inter-param constraints** introduced by Table 7.
- **DD-03 Architecture for I/O:**
  Current: Single-threaded Tkinter app; file-based params and simulated

egram.
Change: Introduce a **`SerialLink` module** that runs on a **worker thread** (or uses Tk `after()` with non-blocking reads). Use a **thread-safe queue** to post events (RX frames, errors) to the UI.

- **DD-04 Protocol Framing & Types:**
  Current: N/A.
  Change: Define **wire protocol** (message headers, payloads, checksums/CRC, endianness, signedness, scaling factors). Decide **float vs fixed-point** for amplitude/sensitivity; document mapping.
- **DD-05 Plotting:**
  Current: Tk Canvas for static preview from JSON.
  Change: Keep Tk Canvas for simplicity **or** switch to `matplotlib` (embed in Tk) for smooth **live streaming**. Ensure **no UI freeze** and bounded memory.
- **DD-06 Storage:**
  Current: `users.json`, `*_params.json`, `egrams/*.json`, `last_device.json`.
  Change: Add `verifications/*.json` (sent vs readback records), **optional SQLite** later if needed.
- **DD-07 Error Handling & UX:**
  Current: Message boxes + status label.
  Change: Centralize errors in a **Comm Log** with levels (INFO/WARN/ERROR). Add **retry** and **Reconnect** controls; keep visual badges in status bar.

3.3 Module Description

Pacemaker State Machine SubSystem

- **Purpose:** Implements permanent-state pacing behavior across AOO, VOO, AAI, and VVI states.
- **Key Functions:**
  - Computes pacing intervals based on LRL and URL.
  - Executes asynchronous pacing for AOO and VOO.
  - Implements inhibited pacing using atrial or ventricular sensing in AAI and VVI .
  - Generates pacing pulses of programmable width and duration.

- **Internal State Variables:** Timing counters, pulse-width timers, flags indicating sense detection.
- **Interactions:** Consumes programmable parameters and sensing inputs; outputs pacing control signals.

Microcontroller Parts SubSystem (Hardware Hiding Layer)

- **Purpose:** Translates abstract pacing and sensing control signals into specific GPIO pin assignments.
- **Key Functions:**
  - Maps ATR_PACE_CTRL, VENT_PACE_CTRL to hardware pin blocks.
  - Routes ATR_CMP_DETECT, VENT_CMP_DETECT from hardware front-end to the controller.
- **State/Global Variables:** None; behaves as a combinational interface.
- **Interactions:** Directly interfaces with board digital I/O; isolates Stateflow from hardware changes.

DCM

- **Purpose:** Provides GUI for doctors in OR and user outside medical setting to manually control state of pacemaker and relay reports of user cardiac data to medical professionals when needed
- **Key Functions:**
  - Allows users to login to their account and view patient specific data such as medical and cardiac reports, active status of device and , and dates of previous medical appointments or emergencies.
  - Select specific modes of operation/states the pacemaker should be in at any given time (ex. AOO, VOO, AAI, VVI).
  - Can modify constants of operation within the state flow design, such as LRL and URL, expected pulse width and amplitudes
- **State/Global Variables:** None currently; will manage stored parameter presets or user session configurations in later deliverables.

- **Interactions:** Will interface with the Pacemaker State Machine to supply updated parameters. May also receive telemetry output from the State flow chart for display or logging.

3.4 Testing

Document test cases for each module. Each test case should include:

1. Purpose of the test
2. Input conditions
3. Expected output
4. Actual output
5. Result (Pass/Fail)

Simulink State Flow Testing

Purpose	Input	Expected Output
Test AOO pacing	Mode AOO, no inputs	ATR_PACE_CTRL outputs pulses at correct timing, board light flashes accordingly
Test VOO pacing	Model VOO, no inputs	VENT_PACE_CTRL outputs pulses at correct timing, board light flashes accordingly
Test AAI	Model AAI, atrial sensing (pulse, period)	No atrial pace is triggered
Test VVI	Model VVI, ventricle sensing (pulse, period)	No ventricular pace is triggered

DCM Testing

ID	Purpose	Input / Setup	Expected Output	Actual Output	Result
T-01	**User registration and authentication**	Register a new user "test1"; attempt duplicate registration and login with correct/incorrect passwords	New user created successfully; duplicate rejected; correct password logs in; wrong password shows red error	Works as expected — login transitions to DCM, duplicates blocked	Pass
T-02	**Parameter range validation and save**	Load `sample_params.json`, edit LRL = 20 (below 30 bpm) and URL = 300 (above 175 bpm)	Error dialogs appear: "Out of range [30–170 bpm]"; parameters not saved	Both validation messages displayed; no file overwritten	Pass
T-03	**Device detection and status feedback**	Load `device_A_params.json`, then select `device_B_params.json`	Pop-up warns: "Different pacemaker detected"; status changes to Connected (green) / Disconnected (gray) appropriately	Prompt and color indicators function correctly	Pass
T-04	**Egram snapshot and preview**	Click "Save Egram Snapshot", then "List Egrams → Preview Selected"	New file appears in `/egrams`; preview shows two traces (blue = Atrial, red = Ventricular)	Snapshot JSON created; preview renders waveform correctly	Pass
T-05	**Parameter file save and**	Edit URL = 130 bpm; click "Save to File"; reopen DCM and reload file	Updated JSON reflects new value; Comm Log: "Saved parameters to	File saved with correct data; log	Pass

persis tence	device_A_params.json ”	and popup confirm ed

3.5 GenAI Usage

GenAI was used for the formatting of this document