Deliverable 1

MECHTRON 3K04

Group #20

October 27, 2025

Table of Contents

1. Group Members

Johnson Ji	jih21	400499564
Arian Nosrati Nik	nosratia	400515273
Hongliang Qi	qih25	400493278
Kelby To	tok13	400507403
Ryan Wang	wang1027	400513220

2. Part 1

2.1 Introduction

2.1.1 Purpose

The purpose of a pacemaker is to regulate and restore a normal heart rhythm in patients with cardiac disorders such as arrhythmias, bradycardia, and heart failure. The pacemaker accomplishes this by delivering small electrical pulses to the atria and ventricles to make the heartbeat at the correct speed and pattern.

The system consists of two main components: the Pacemaker and the Device Controller-Monitor (DCM). The Pacemaker handles sensing and pacing functions, while the DCM allows users to configure, monitor, and manage the settings of the Pacemaker.

2.1.2 Goals

The main goal of Deliverable 1 is to design and implement the foundational components of the Pacemaker and DCM. Specifically:

Pacemaker: Create state flow models for AOO, VOO, AAI, and VVI modes with parameters specified in the Deliverable 1 document.

DCM: Develop an interface that enables user registration and login, displays pacing modes, and allows input and storage of modifiable parameters.

Documentation: Provide a detailed document outlining the design process, decisions, implementation, and testing procedures for the Pacemaker and DCM.

2.1.3 Scope

Deliverable 1 focuses on developing the initial components of the Pacemaker and DCM. This includes:

Creating state flow models for the AOO, VOO, AAI, and VVI pacing modes.

Building the DCM interface with registration, login, and parameter customization functionality.

Documenting the design and implementation process.

2.2 Requirements

2.2.1 Modes

Mode	Pacing	Sensing	Response to Sensing	Behaviour
AOO	Atrium	None	None (asynchronous)	Delivers atrial pacing pulses at a fixed rate, regardless of intrinsic activity
VOO	Ventricle	None	None (asynchronous)	Delivers ventricular pacing pulses at a fixed rate, regardless of intrinsic activity
AAI	Atrium	Atrium	Inhibited	Paces the atrium only if no intrinsic atrial activity is sensed within the programmed interval
VVI	Ventricle	Ventricle	Inhibited	Paces the ventricle only if no intrinsic ventricular activity is sensed within the programmed interval

2.2.2 DCM behavior Requirements

This section also outlines the functional requirements for DCM. The DCM must be capable of performing Log In, Log Out, Sign Out, and Registration operations, and it must be able to store data for up to 10 users. The DCM must possess a clear and straightforward main interface, which should include a secondary menu for modifying parameters and modes, as well as for viewing electrocardiograms (EGram). For the D1 unit, it must correctly modify and store the corresponding parameters, and it must reserve the necessary electrocardiogram parameters to accommodate future D2 requirements. A help window is also needed to help the doctor to understand the essential modes and parameters. The welcome windows also should display the state of the pacemaker (ie. Connected, disconnected, noise, new device, etc.).

2.3 Design

2.3.1 System Architecture

The Simulink system consists of 3 important modules: Input Hardware Hiding, Main, and Output Hardware Hiding.

Input Hardware Hiding

The purpose of Input Hardware Hiding is to convert the inputs to the system to usable variables in the software. The inputs to this module are the programmable parameters for the pacemaker and digital read signals from the heart, and the outputs are connected to the Main module's input. In this module, the inputs are limited to their respective acceptable ranges using saturation functions. This module also calculates unit conversions when applicable and provides signal conditioning for reliable sensing.

Main

The Main module consists of 4 submodules, one for each of the 4 modes: AOO, VOO, AAI, VVI, as well as an idle state. The inputs to this module are the outputs from the Input Hardware Hiding module, where each parameter has been processed to be acceptable for the software. This module acts as the high-level control logic for the pacemaker, where the mode is chosen according to the "Mode" programmable parameter. In each submodule, there are charging and pacing states that form a cycle, as well as additional idle and sensing states for the AAI and VVI submodules. These submodules control the output variables that are sent to the pacemaker via the Output Hardware Hiding module.

Output Hardware Hiding

The Output Hardware Hiding module takes in the outputs of the Main module as inputs and convert them to usable signals for the pacemaker hardware. This includes outputting digital signals using digital write functions, as well as analog signals using PWM output functions.

2.3.2 Programmable parameters

The pacemaker's operation is governed by a set of adjustable parameters that define its behavior. These parameters are outlined in the following table:

Parameter	Description	Range	Units
Amplitude	The electrical voltage of the pacing pulse. This controls the atrial or ventricular channels, depending on the selected mode.	0.5 ~ 5	V
Sensitivity	The minimum level of intrinsic electrical signal that the pacemaker can detect. A lower value indicates higher sensitivity.	0.25 ~ 10	mV
Lower Rate Limit (LRL)	The minimum rate at which the pacemaker will pace the heart in the absence of intrinsic activity.	30 ~ 175	BPM
Upper Rate Limit (URL)	The maximum rate at which the pacemaker is permitted to pace.	50 ~ 180	BPM
Pulse Width	The duration of the electrical pacing pulse. This determines the time over which the voltage is applied.	0.05 ~ 1.9	ms
Atrial Refractory Period (ARP)	The period after an atrial event during which the atrial channel ignores any new input.	150 ~ 500	ms
Ventricle Refractory Period (VRP)	The period after a ventricular event during which the ventricular channel ignores any new input.	150 ~ 500	ms
Mode	The operational pacing mode of the device. (AOO = 0, VOO = 1, AAI = 2, VVI = 3)	0, 1, 2, 3	-

2.3.3 Hardware inputs and outputs

Hardware Inputs (Sensed Signals)

The inputs are electrical signals detected from the heart. These signals are:

Signal Name	Pin	Description	Range
ATR_CMP_DETECT	D0	The electrical activity of the atrium. This signal detects pulses in the atrium for the pacemaker to pace accordingly	0/1
VENT_CMP_DETECT	D1	The electrical activity of the ventricle. This signal detects pulses in the ventricle for the pacemaker to pace accordingly.	0/1

Hardware Outputs (Controlled Signals)

The outputs are electrical pulses generated by the pacemaker to stimulate the heart. The microcontroller generates precise digital triggers that control the pacing circuitry.

Signal Name	Type	Pin	Description
Pacing Circuit Control			
PACE_GND_CTRL	Digital	D10	Controls the grounding path for the pacing circuit.
PACE_CHARGE_CTRL	Digital	D2	Enables the charging of the capacitor used for the pacing pulse.
PACING_REF_PWM	PWM	D5	Pulse-Width Modulation signal that regulates the voltage amplitude of the pacing pulse.
Atrium Channel Control			
ATR_GND_CTRL	Digital	D11	Controls the grounding path for the atrial output channel.
ATR_PACE_CTRL	Digital	D8	Controls current flow to discharge the capacitor for pacing atrium
ATR_CMP_REF_PWM	PWM	D6	Sets the reference voltage for the atrial sensing comparator (controls atrial sensitivity).

Ventricle Channel Control			
VENT_GND_CTRL	Digital	D12	Controls the grounding path for ventricular output channel.
VENT_PACE_CTRL	Digital	D9	Controls current flow to discharge the capacitor for pacing ventricle
VENT_CMP_REF_PWM	PWM	D3	Sets the reference voltage for the ventricular sensing comparator (controls ventricle sensitivity).
System Control & Status			
FRONTEND_CTRL	Digital	D13	Enables/Disable the analog sensing circuitry.
RED_LED	Digital	RED_LED	A status indicator for atrium pacing modes
BLUE_LED	Digital	BLUE_LED	A status indicator for ventricle pacing modes

2.3.4 State machine design

The core logic of the pacemaker is implemented through a set of finite state machines (FSMs). Each pacing mode has a dedicated FSM that dictates how the system responds to timed and/or sensed events. The state machines for the four implemented modes (AOO, VOO, AAI, VVI) are described below.

AOO

AOO mode paces the atrium at a fixed rate, without sensing or response to atrial activities

Current State	Trigger	Actions	Next State
AOO_Entry	After completing all actions	Set pacing voltage Turn off all ventricle controls	AOO_Charging
AOO_Charging	after (lrl_ms – pulse_width) time in ms	Stop discharging capacitor to atrium Charge Capacitor	AOO_Pacing
AOO_Pacing	After (pulse_width) time in ms	Stop charging capacitor Discharge capacitor to atrium	AOO_Charging

VOO

VOO mode paces the ventricle at a fixed rate, without sensing or response to ventricular activities

Current State	Trigger	Actions	Next State
VOO_Entry	After completing all actions	Set pacing voltage Turn off all atrium controls	VOO_Charging
VOO_Charging	after (lrl_ms – pulse_width) time in ms	Stop discharging capacitor to ventricle Charge Capacitor	VOO_Pacing
VOO_Pacing	After (pulse_width) time in ms	Stop charging capacitor Discharge capacitor to ventricle	VOO_Charging

AAI

AAI mode senses the atrium and inhibits atrial pacing if an intrinsic atrial event is detected within the LRL period.

Current State	Trigger	Actions	Next State
AAI_Entry	After completing all actions	Set pacing voltage Turn off all ventricle controls Stop discharging capacitor to atrium Charge Capacitor	AAI_Idle
AAI_Idle	Case 1: Heartbeat detected in atrium	Turn off Red LED	Case 1: AAI_Sensing
	Case 2: Heartbeat not detected after (lrl_ms – pulse_width) time in ms		Case 2: AAI_Pacing
AAI_Detected	After (arp) time in ms	Set local variable “heartbeat_detected” to 1	AAI_Idle
AAI_Pacing	After (pulse_width) time in ms	Stop charging capacitor Discharge capacitor to atrium Turn on Red LED	AAI_Charging
AAI_Charging	After completing all actions	Stop discharging capacitor to atrium Charge Capacitor	AAI_Idle

VVI

VVI mode senses the atrium and inhibits atrial pacing if an intrinsic atrial event is detected within the LRL period.

Current State	Trigger	Actions	Next State
VVI_Entry	After completing all actions	Set pacing voltage Turn off all atrium controls Stop discharging capacitor to ventricle Charge Capacitor	VVI_Idle
VVI_Idle	Case 1: Heartbeat detected in atrium	Turn off Blue LED	Case 1: VVI_Sensing
	Case 2: Heartbeat not detected after (lrl_ms – pulse_width) time in ms		Case 2: VVI_Pacing
VVI_Detected	After (vrp) time in ms	Set local variable “heartbeat_detected” to 1	VVI_Idle
VVI_Pacing	After (pulse_width) time in ms	Stop charging capacitor Discharge capacitor to ventricle Turn on Blue LED	VVI_Charging
VVI_Charging	After completing all actions	Stop discharging capacitor to ventricle Charge Capacitor	VVI_Idle

2.3.5 Simulink diagram

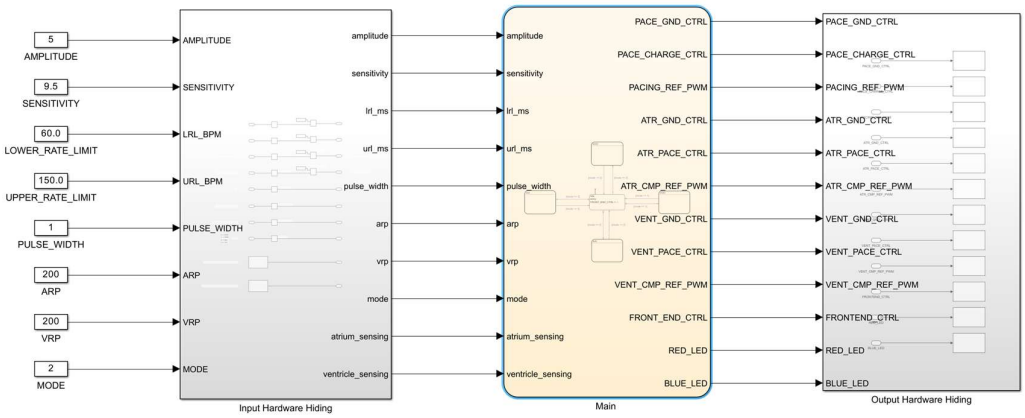


Figure 1. Full Simulink Model

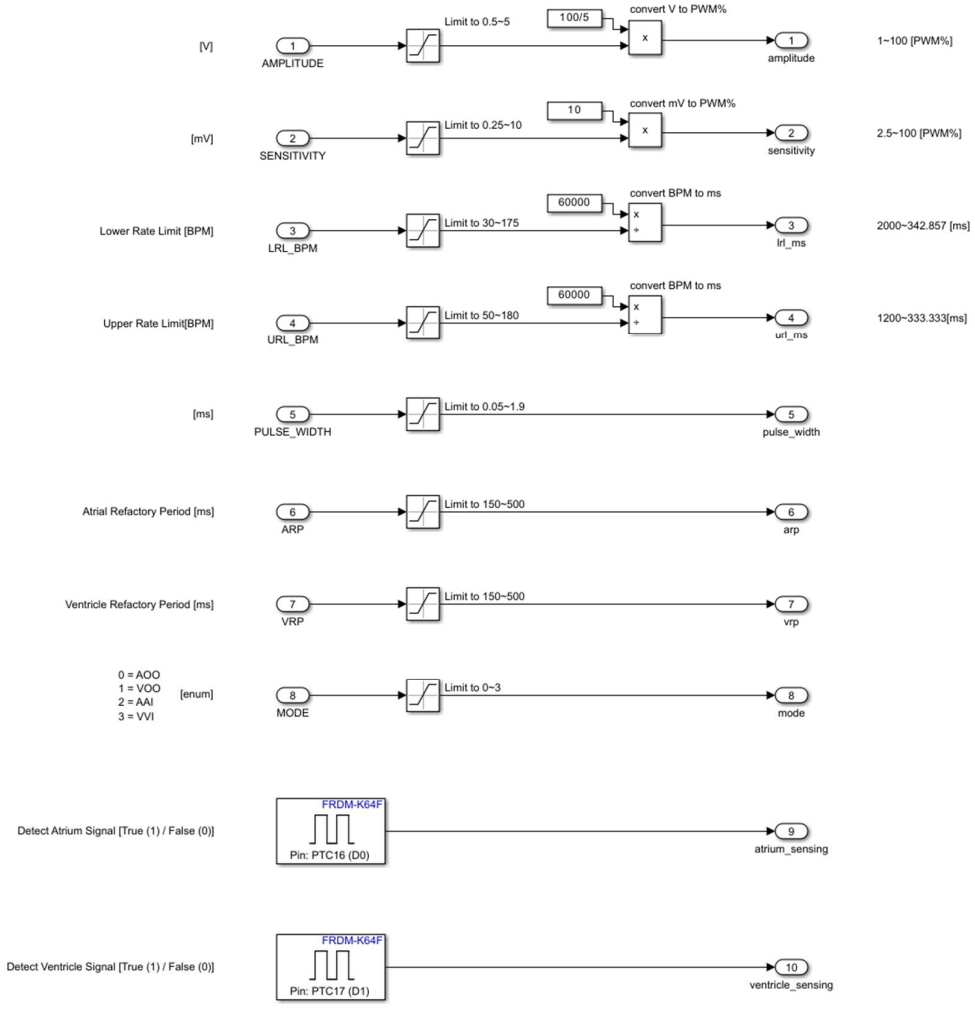


Figure 2. Input Hardware Hiding Module

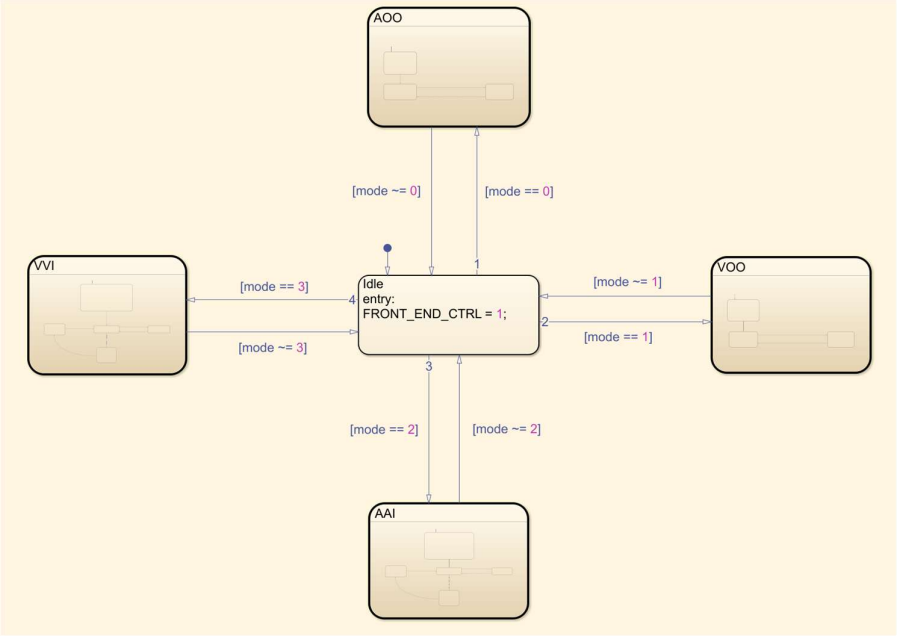


Figure 3. Main Module

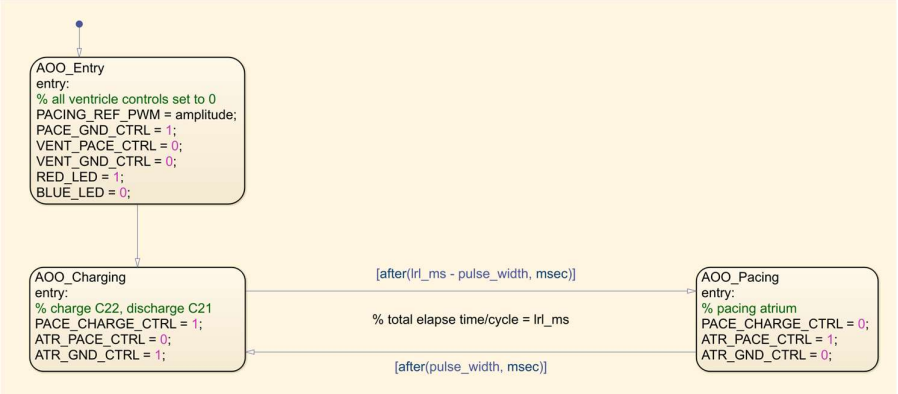


Figure 4. AOO Submodule in Main

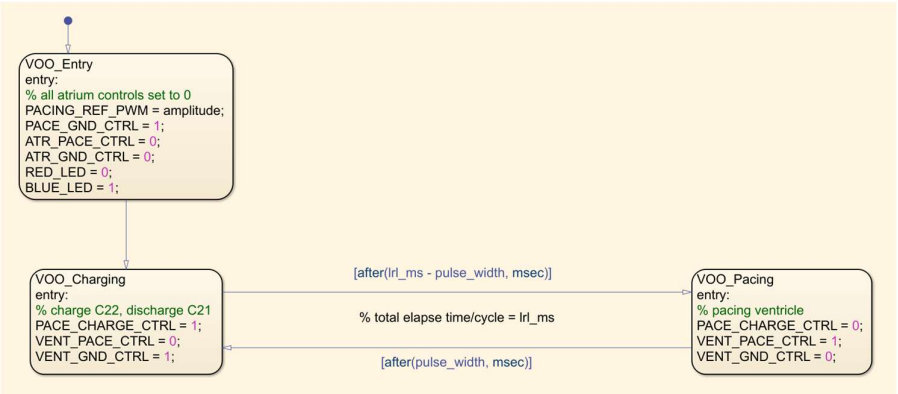


Figure 5. VOO Submodule in Main

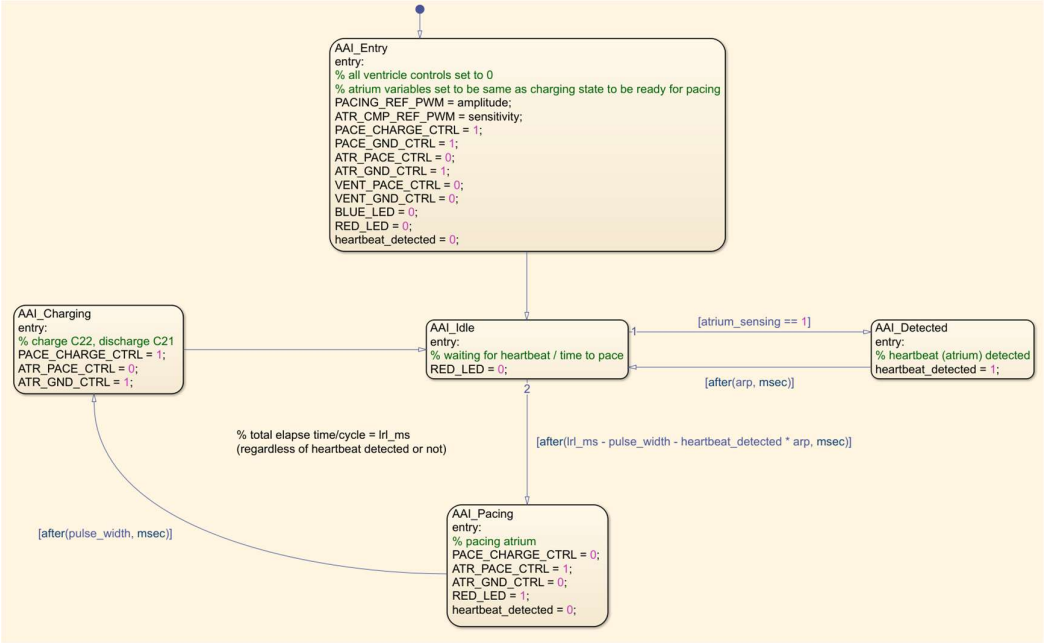


Figure 6. AAI Submodule in Main

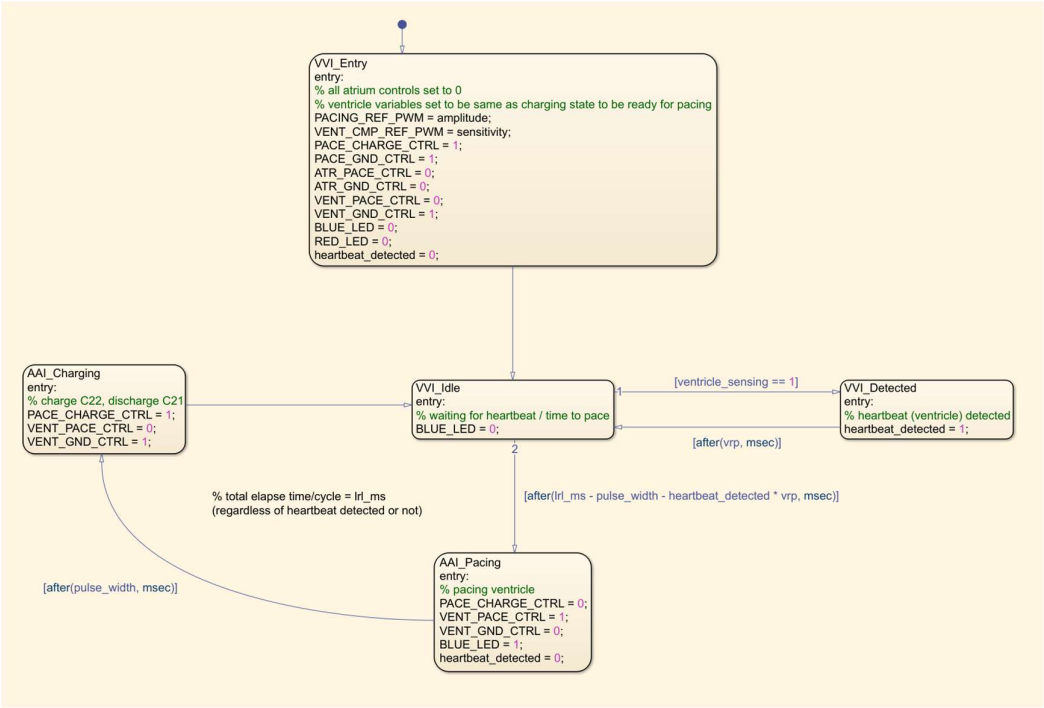


Figure 7. VVI Submodule in Main

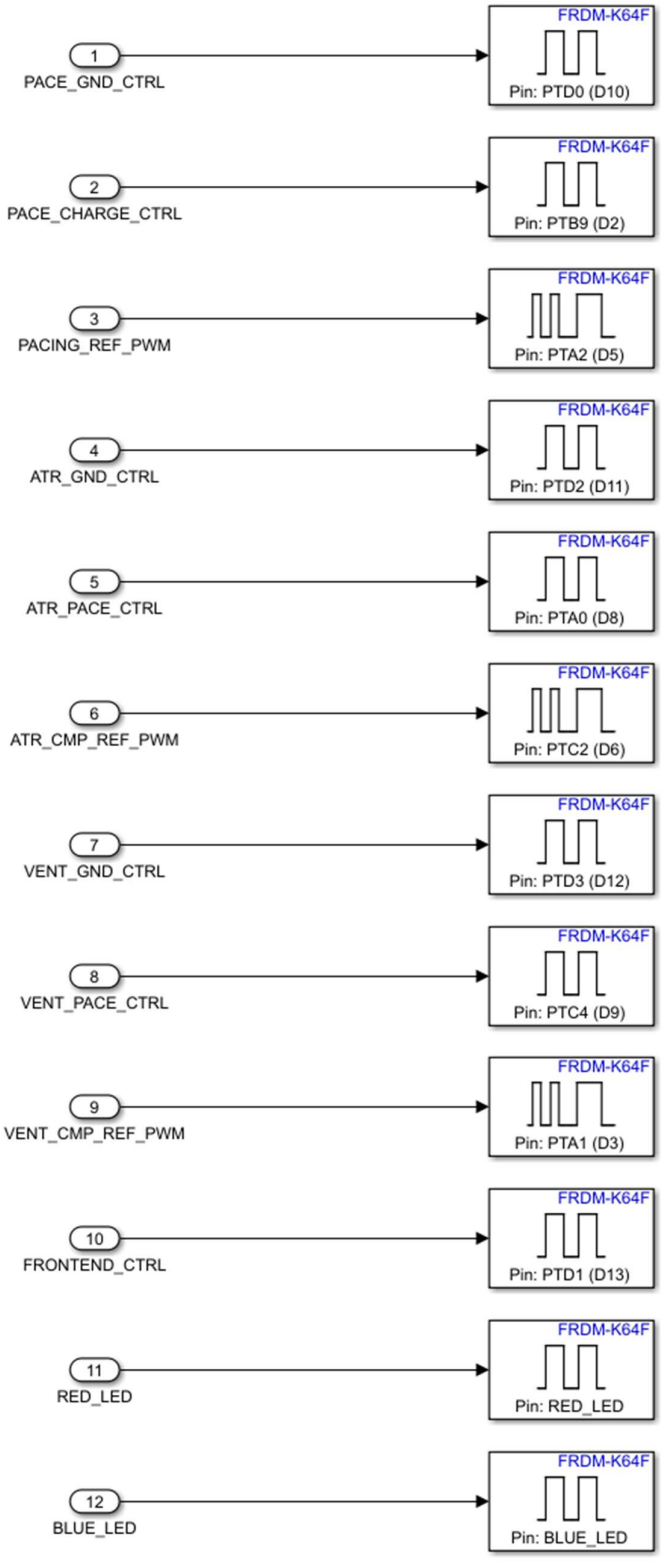


Figure 8. Output Hardware Hiding Module

2.3.6 DCM Overview & System Architecture

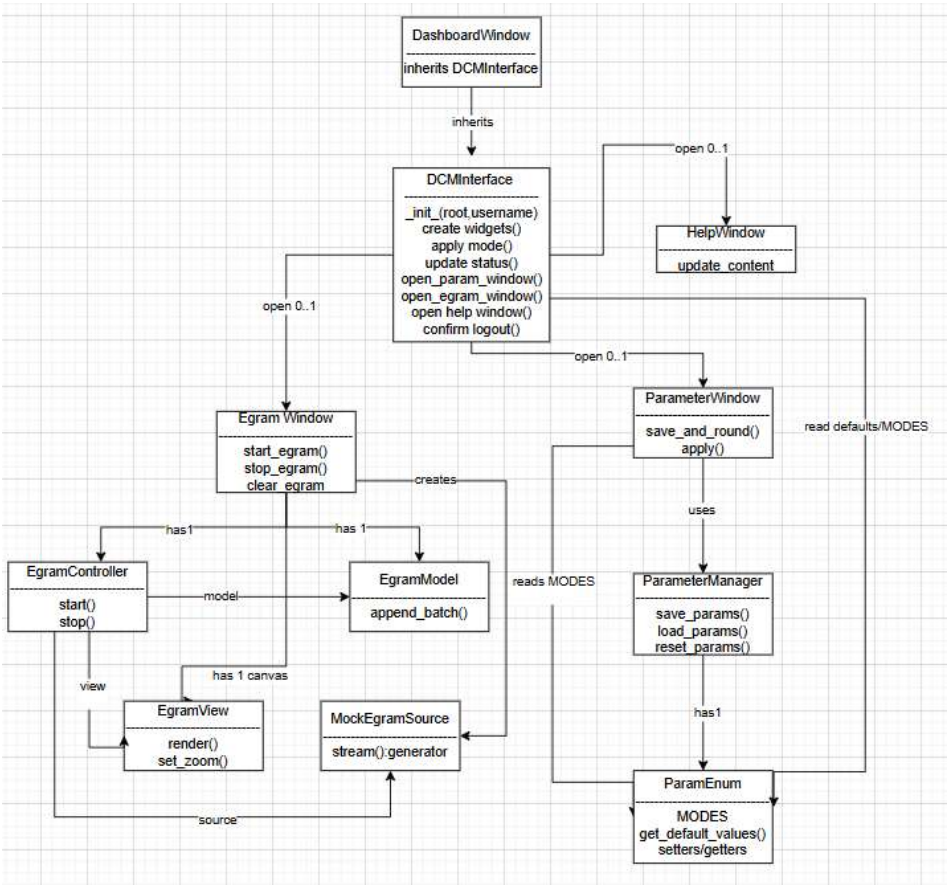
The Device Control Module (DCM) is a desktop UI implemented with Tkinter. The navigation flow is App → Dashboard → {Parameter Window, Help Window, Egram Window}.

- Dashboard provides entry points to parameter editing, help documents, and egram preview.
- Parameter Window edits and persists programmable parameters with strict validation and step rounding.
- Help Window render local docs for modes/parameters with a graceful fallback when files are missing.
- Egram Window visualizes (mock) atrial/ventricular signals in D1; in D2 the data source will be replaced by a serial/real source.

Module	Purpose / Responsibility
auth.py	Manages user registration, login, and logout
dashboard.py	Main post-login hub. Provides access to mode selection, parameter editing, EGram viewing, and logout.
mode_config.py	Defines pacing modes and programmable parameters through the ParamEnum class.
ParamOps.py	Implements the ParameterWindow for parameter input, rounding, validation, and JSON persistence.
EGdiagram.py	Handles EGram visualization and control (Start, Stop, Clear) with safety warnings.
Help_Window.py	Displays contextual help content; gracefully handles missing help files.

Communication.py	Not complete in D1 but will be finished in D2 to apply the serial communication with the pacemaker board.
main.py	Launches the overall DCM and initializes UI windows.

Class diagram



Has 1 marks composition, the parent makes and strictly owns that part, like DCM interface to ParameterManager, EgramWindow to Model, Controller/View. But these are all only one thing, and it is tied to the parent. When the parent is destroyed, so are the other parts. By contrast, open 0..1 marks aggregation for optional, single-instance child windows (ParameterWindow,EgramWindow,HelpWindow). They are made in a lazy fashion, can be closed independently, and at most one is open at a time. The parent holds a nullable reference and simply brings the window back into reality or creates it as necessary.

Relational Hierarchy:

```
App (main.py)
└─ DashboardWindow  [opened by App after login]
   ├─ ParameterManager
   │  └─ ParamEnum               [defaults/ranges]
   ├─ ParameterWindow (0..1)     [opened from Dashboard; single-instance]
   ├─ HelpWindow (0..1)          [opened from Dashboard; single-instance]
   └─ EgramWindow (0..1)         [opened from Dashboard; single-instance]
      ├─ EgramModel              [composition]
      ├─ EgramView               [composition]
      └─ EgramController (0..1)  [managed by EgramWindow: start/stop]
```

Uses Relationship:

```
graph TD
    %% Main app layer
    A[main.py] --> B[dashboard.py]
    A --> H[auth.py]

    %% Dashboard dependencies
    B --> C[Help_Window.py]
    B --> D[EGdiagram.py]
    B --> E[ParamOps.py]
    B --> G[mode_config.py]  %% indirectly via ParamOps

    %% EGdiagram internal structure
    D --> D1[EgramController (internal thread)]
    D --> D2[EgramView (canvas)]

    %% Help_Window dependencies
    C --> F[Load_JSON.py]   %% JSON loader module

    %% Parameter manager dependency
    E --> G[mode_config.py]

    %% Authentication standalone
    H --> |used by| A
```

2.3.7 Screens & Responsibilities

- Welcome/Login: register/login (up to 10 local users) and then route to Dashboard.
- Dashboard: main menu; shows mode presentation and opens sub-windows.
- Parameter Window: mode selection (AOO/VOO/AAI/VVI), parameter editing, Save/Load/Reset with validation & rounding; immediate UI refresh after Load/Reset.
- Help Window: renders mode/parameter descriptions from local JSON; if missing, displays a user-friendly fallback message.
- Egram Window: start/stop/clear plotting; guards against double start and disallow clear while running.

2.3.8 Programmable Parameters (specification)

All parameters have persisted locally; invalid input is rejected with a clear message; valid input is snapped to the nearest legal step before saving.

Parameter	Range	Step	Default	Unit	Modes	Validated in (example)
Lower Rate Limit (LRL)	30–175	**1** within 50–90, else **5**	60	ppm	All	`set_lower_rate_limit()`
Upper Rate Limit (URL)	50–175	**5**	120	ppm	All	`set_upper_rate_limit()`
Atrial Amplitude (AA)	0.5–3.2 (**0.1**); 3.5–7.0 (**0.5**)	piecewise (see left)	3.5	V	AOO/ AAI	`set_atrial_amplitude()`
Atrial Pulse Width (APW)	**0.05** OR 0.10–1.90	**0.10**	0.4	ms	AOO/ AAI	`set_atrial_pulse_width()`
Ventricular Amplitude (VA)	0.5–3.2 (**0.1**); 3.5–7.0 (**0.5**)	piecewise	3.5	V	VOO/ VVI	`set_ventricular_amplitude()`
Ventricular PW (VPW)	**0.05** OR 0.10–1.90	**0.10**	0.4	ms	VOO/ VVI	`set_ventricular_pulse_width()`
ARP	150–500	**10**	250	ms	AAI	`set_arp()`
VRP	150–500	**10**	250	ms	VVI	`set_vrp()`

2.3.9 Validation & Rounding Rules (summary)

Lower Rate Limit (LRL)

- Valid range: [30, 175] bpm
- Stepping:
  - $30 \leq x < 50$: step 5 bpm
  - $50 \leq x \leq 90$: step 1 bpm
  - $90 < x \leq 175$: step 5 bpm
- Rounding: rounded to the nearest allowed step within the relevant sub-range, then clamped to [30, 175].
- Cross-constraint: LRL ≤ URL (if Upper_Rate_Limit already set). Violations raise ValueError.
- Stored as: int.

Intent: fine control in the normal clinical window (50–90), coarser control at lower/higher extremes.

Upper Rate Limit (URL)

- Valid range: [50, 175] bpm
- Stepping: uniform step 5 bpm across the whole range.
- Rounding: to nearest 5 bpm, then clamped to [50, 175].
- Cross-constraint: URL > LRL strictly. Violations raise ValueError.
- Stored as: int.

Intent: enforce a clean upper bound with coarse but standard pacing granularity.

Atrial Amplitude (AA)

- Valid ranges (two disjoint bands):
  - 0.5–3.2 V → step 0.1 V
  - 3.5–7.0 V → step 0.5 V
- Rounding: to the nearest step within the band; values in the gap (3.2, 3.5) are invalid.
- Stored as: float.

Intent: provide fine resolution at lower amplitudes, coarser at high outputs; explicitly disallow the discontinuity gap.

Ventricular Amplitude (VA)

- Valid ranges (two disjoint bands):
  - 0.5–3.2 V → step 0.1 V
  - 3.5–7.0 V → step 0.5 V
- Rounding: 0.5–3.2 V uses round(x*10)/10; 3.5–7.0 V uses _round_to_step(x, 0.5).
  Gap (3.2, 3.5) is invalid.
- Stored as: float.

Intent: identical stepping policy as atrial amplitude.

Atrial Pulse Width (APW)

- Valid values:
  - exactly 0.05 ms, or
  - 0.10–1.90 ms with step 0.10 ms
- Rounding:
  - 0.05 is accepted only when essentially equal to 0.05 (epsilon check).
  - 0.10–1.90 ms values are snapped to 0.1-ms grid with round((x-0.10)/0.10)*0.10 + 0.10, then round(..., 2).
- Stored as: float with 2-decimal rounding.

Intent: special “minimal” pulse option at 0.05 ms and a standard 0.1-ms grid for the typical operating range.

Ventricular Pulse Width (VPW)

- Valid values:
  - exactly 0.05 ms, or
  - 0.10–1.90 ms with step 0.10 ms
- Rounding: same policy as APW.
- Stored as: float with 2-decimal rounding.

Intent: mirror of atrial pulse-width behavior.

ARP (Atrial Refractory Period)

- Valid range: [150, 500] ms
- Stepping: 10 ms
- Rounding: _round_to_step(x, 10), stored as int.

Intent: discrete timing grid consistent with clinical configuration increments.

VRP (Ventricular Refractory Period)

- Valid range: [150, 500] ms
- Stepping: 10 ms
- Rounding: _round_to_step(x, 10), stored as int.

Intent: identical to ARP.

get_Atrial_Amplitude() / get_Ventricular_Amplitude() return 0 if the stored value is numerically 0 (safety-oriented guard), else return the stored float.

get_default_values() aggregates the current internal values (after any rounding/validation).

3. Part 2

3.1 Requirements & Potential Changes

DCM

Module	Requirements	Potential Changes / Revisions
HelpWindow	Display helps documentation for parameters and pacing modes, with navigation and formatted text.	Adding new help topics (e.g., D2 modes), support for multimedia content, or online help updates.
ParamEnum	Store pacemaker modes and parameters, provide getter and setter interfaces with validation and stepping rules.	Adding new pacing modes (D2, AOOR, etc.), new parameters for advanced therapy modes.
ParameterManager	Manage parameter operations: save, load, reset, and apply; GUI to edit parameters based on mode.	Improve validation rules, support bulk import/export, additional GUI elements for future parameters.
WelcomeWindow	Provide login and registration functionality; launch dashboard upon successful login.	Integration with secure authentication APIs, multi-user support, and potential cloud-based storage.
ParameterWindow	Edit pacing parameters per mode; Save/Load/Reset; immediate UI refresh; enforce cross-constraints and step rounding.	Add presets/bulk import–export; inline validation tips; support future modes/extra fields without UI rewrites.
Egram	Start/Stop/Clear; live plotting with mock source (D1); guard double-Start; disable Clear while running.	Swap to serial adapter; add channel(D1) filters/zoom/export (PNG/CSV); performance throttling and annotations.
DashboardWindow	Navigation hub; open/lift single instances of Help or Egram or Parameter windows; show session/status.	Role-based buttons; quick mode/parameter presets; session banner and status toasts.

App	Initialize Tk; drive Login → Dashboard; graceful shutdown (no stray threads).	Packaging (installer), logging/error recovery, start-up checks (data dir/files).
auth.py	Local register or login (≤10 users); duplicate prevention; user feedback dialogs.	Stronger hashing/salt, multi-user beyond cap, secure authentication APIs or cloud/DB store.

Simulink

Module	Requirements	Potential Changes / Revisions
Input hardware hiding module	Maps physical pins to logical signals for use in Simulink. Shields main design from hardware changes.	Update for new pin mappings, hardware upgrades, or added input devices.
Main module	Controls selection and logic for all pacing modes. Coordinates programmable parameters and signal flow.	Add new pacing modes, expand parameter sets, or refactor for modularity.
AOO Submodule in main	Delivers fixed-rate pulses to the atrium, ignoring sensed events. Uses amplitude, pulse width, and interval parameters.	Add diagnostics, adjust timing, or prepare for AOOR extension.
VOO Submodule in main	Send fixed-interval pulses to the ventricle, and independent of sensing. Parameterized by amplitude and pulse width.	Integrate safety checks, change timing, and support for VOOR.
AAI Submodule in main	Paces atrium only if no natural event is sensed within interval. Implements sensing and inhibition logic.	Tweak sensing, add rate adaptation, or upgrade for AAIR.
VVI Submodule in main	Stimulates ventricle only if no intrinsic beat is detected within window. Uses inhibition of logic and ventricular sensing.	Enhance sensing, support VVIR, or add error handling.
Output Hardware Hiding module	Maps logic outputs to physical pins for actuation. Isolates main logic from hardware dependencies.	Update routing for new hardware, integrate confirmation feedback, or add outputs.

3.2 Design Decision & Potential Changes

DCM

Module	Design Decision	Potential Changes / Revisions
HelpWindow	Tkinter-based GUI; JSON files for content; text widget formatting.	Consider switching to web-based help, dynamic content loading, or modular UI frameworks.
ParamEnum	Python class with validated getters or setters; uses constants for mode definitions.	Might adopt database-driven parameters, use enums for clarity, or external config files for scalability.
ParameterManager	GUI-bound parameter management; state tracking; method resolution via string names.	Refactor to MVC pattern, improve input validation, or integrate real-time device updates.
WelcomeWindow	Simple Tkinter GUI for login/registration; dashboard launch.	Upgrade authentication security, support OAuth, and redesign GUI for modern UX.
ParameterWindow	Mode-aware form; calls ParamEnum setters to validate and step-round; immediate UI refresh on Save or Load or Reset; disables irrelevant fields per mode.	Presets and bulk import/export; inline hints for invalid inputs; undo/redo; add fields for future modes without UI rewrites
Egram	MVC split: Window = controls; View = canvas render; Controller = acquisition thread (mock source in D1); redraw throttled; double-Start guarded; Clear disabled while running.	Swap to serial adapter (D2) without UI changes; add filters/zoom/export (PNG/CSV); performance tuning; annotations/cursors.
DashboardWindow	Navigation hub; single-instance create-or-lift for Help/Egram/Parameter; lightweight session status.	Role-based buttons; quick presets; status toasts; non-blocking open/close.

App	Tk app lifecycle owner; show Login → open Dashboard; graceful shutdown (no stray threads).	Packaging/installer; structured logging & error recovery; startup checks (data dir/files).
auth.py	JSON-backed local accounts (≤10 users); duplicate prevention; stores password hashes (not plaintext).	Stronger hashing/salt & account lockouts; OAuth/SSO; multi-user beyond cap; optional cloud/DB store.

Simulink

Module	Requirements	Potential Changes / Revisions
Input hardware hiding module	Decouple hardware details from Simulink logic. Use abstraction to support hardware changes without modifying main control logic.	Adapt for new boards, sensors, or pin mappings. Add preprocessing to for example like noise filtering.
Main module	Centralize control/selection of pacing mode. Parameterize for easy updates.	Expand with new pacing modes. Integrate more complex parameter sets.
AOO Submodule in main	Deliver fixed-rate atrial pulses, no sensing logic	Enable diagnostics. Support for rate-responsive (AOOR) upgrades.
VOO Submodule in main	Fixed-rate ventricular pacing, no sensing.	Improve safety diagnostics. Add VOOR support.

AAI Submodule in main	Only pace if no intrinsic atrial event detected. Include sensing and inhibition logic.	Add rate-adaptive logic (AAIR). Refine sensing algorithms.
VVI Submodule in main	Only pace ventricle if no intrinsic event detected. Implement sensing/refractory logic.	Support for VVIR. Enhance error handling and diagnostics.
Output Hardware Hiding module	Abstraction for output mapping to physical pins.	Accommodate new output hardware. Add feedback/confirmation or expandable outputs.

3.3 Module Description

3.3.1 Module Guide

App (main.py)

- Objective: To set up the Tk app, display login, Dashboard, manage the lifetime. Key functions/methods:
- – Public: run, _show_login, _open_dashboard.
- – Internal: maintain root loop, handle the root loop, window refs.
- Global/state variables such as root, _login_win, _dashboard.
- Interactions: Use WelcomeWindow/auth to login; open DashboardWindow.

WelcomeWindow.

- Purpose: Log credentials and redirect to Dashboard on success. Key functions/methods:
- – Public: show, on_register, on_login.
- – Internal: hash password, input checks.
- Global/state variables: name entry, pass entry, visible flag.
- Interactions: calls auth.register_user / auth.login_user; notifies App.

DashboardWindow (dashboard.py).

- Purpose: UI hub; lift/lift single instances of child windows. Key functions/methods:
- – Public: open_help_window, open_egram, open_parameter_window.
- – Internal: single-instance window management.
- Global/state variables: username, help_window, egram_window, param_window.
- Interactions: Creating/raising HelpWindow, EgramWindow, ParameterWindow.

HelpWindow (Help_Window.py).

- Purpose: To render help docs for modes/parameters with graceful fallback. Key functions/methods:
- – Public: update_content, load help content.
- – Internal: display param document, display mode document, display text content.
- Global/state variables: topics dict, current topic, content area widget.

- Interactions: Reads local JSON help files, updating GUI text live.

ParameterWindow.

- Purpose: Edit/save/load/reset pacing parameters by mode with validation/rounding. Key functions/methods:
- – Public: save, load, reset, enabled_fields.
- – Internal: fill entries, toggle fields by mode.
- Global/state variables: entry widgets, mode var, last snapshot.
- Interactions: Use of ParamEnum for rules; JSON persistence using ParameterManager.

ParamEnum (mode_config.py).

- Purpose: hold ranges/steps/defaults and ensure that validation/cross-constraints are enforced. Key functions/methods:
- – Public: getters (get_), setters (set_), fields_for, validate_cross.
- – Internal: is_number, round_to_step.
- Global/state variables: DEFAULTS, RANGES/STEPS, MODES mapping.
- Interactions: Used by ParameterWindow/ParameterManager for consistent rules.

ParameterManager (ParamOps.py).

- Purpose: Save/load/reset parameter sets; provide defaults. Key functions/methods:
- – Public: save_params, load_params, reset_params.
- – Internal: ensure data dir, atomic write.
- Global/state variables: file path (data/parameters.json), last loaded.
- Interactions: Reads/writes JSON; works with ParamEnum to set defaults.

EgramWindow (EGdiagram.py).

- Purpose: Container of egram controls (Start/Stop/Clear) and view wiring. Key functions/methods:
- – Public: start_egram, stop_egram, clear_egram.
- – Internal: manage button state, schedule redraw.
- Global/state variables: controller ref, view/canvas, running flag (UI).
- Interactions: Commands EgramController; forwards snapshots to EgramView.

EgramView (EGdiagram.py).

- Purpose: Sketch atrial/ventricular traces on a canvas. Key functions/methods:
- – Public: render.
- – Internal: format axes/labels, zoom text update.
- Global/state variables: canvas, last-drawn bounds.
- Interactions: Receives snapshots from controller/model; draws UI.

EgramController (EGdiagram.py).

- Objective: Background acquisition loop and user interface redraw scheduling. Key functions/methods:
- – Public: start, stop, clear, snapshot (optional).
- – Internal: acquisition thread loop, source adapter.
- Global/state variables: running flag, model buffer, sampling rate.
- Interactions: Pulls frames from mock/serial; pushes updates to view through UI scheduler.

auth.py.

- Purpose: Minimal local account store (≤10 users). Key functions/methods:
- – Public: register_user, login_user, logout_account, count_users.
- – Internal: load/save JSON, duplicate/limit checks.
- Global/state variables: users list in memory, file path (data/users.json).
- Interactions: Initiated via WelcomeWindow/App to authenticate users.

3.3.2 MIS — Module Interface Specification

App (main.py).

- Purpose: Lifecycle owner; present login, open Dashboard. Key functions/methods:
- – Public: run(), _show_login(), _open_dashboard(username:str).
- – Internal: keep root loop alive, destroy on exit.
- Global/state variables: root Tk, _login_win, _dashboard.
- Interactions: Uses auth via WelcomeWindow; instantiates DashboardWindow.

WelcomeWindow.

- Purpose: Authentication UI; delegate persistence to auth.py. Key functions/methods:
- – Public: show(), on_register(name:str, pw:str)->bool, on_login(name:str, pw:str)->bool.
- – Internal: hash password, input non-empty checks.
- Global/state variables: name entry, pass entry, visible flag.
- Interactions: Calls auth.register_user/login_user; signals App to open Dashboard.

DashboardWindow.

- Purpose: Hub for child windows.
- – Internal: ensure single-instance (create or lift).
- Global/state variables: help_window, egram_window, param_window, username.
- Interactions: Constructs/raises HelpWindow, EgramWindow, ParameterWindow.

HelpWindow.

- Purpose: Topic-based help viewer with fallback. Key functions/methods:
- – Public: update_content(topic:str)->None, load help content.
- – Internal: display param document, display mode document, display text content.
- Global/state variables: topics, current topic, content area.
- Interactions: Reads data/Param_Help.json, data/Mode_Help.json; updates GUI text.

ParameterWindow.

- Purpose: Mode-aware parameter editor; Save/Load/Reset; immediate refresh. Key functions/methods:
- – Public: save()->bool, load()->dict, reset()->dict, enabled_fields(mode:str)->list[str].
- – Internal: fill entries with normalized values, disable irrelevant fields.
- Global/state variables: entry widgets per field, mode var, last snapshot.
- Interactions: Uses ParamEnum setters and validate_cross; uses ParameterManager JSON I/O.

ParamEnum.

- Purpose: Validation and normalization source of truth. Key functions/methods:
- – Public: get_default_, set_(x)->number, fields_for(mode)->list[str], validate_cross(values)->None.
- – Internal: is_number, round_to_step (piecewise steps).
- Global/state variables: DEFAULTS, RANGES/STEPS, MODES dict.
- Interactions: Called by ParameterWindow/Manager; no file I/O.

ParameterManager.

- Purpose: Persistence for parameters and defaults (no validation). Key functions/methods:
- – Public: save_params(values:dict, mode:str|None)->bool, load_params()->dict, reset_params()->dict.
- – Internal: ensure data dir, atomic write, merge defaults on load.
- Global/state variables: data/parameters.json path, last loaded cache.
- Interactions: Reads/writes JSON; relies on ParamEnum for defaults.

EgramWindow.

- Purpose: Wire Start/Stop/Clear and trigger redraws. Key functions/methods:
- – Public: start_egram()->bool, stop_egram()->None, clear_egram()->None.
- – Internal: guard double-start, disable clear while running.
- Global/state variables: controller ref, view/canvas, running UI state.
- Interactions: Commands EgramController; passes snapshots to EgramView.

EgramView.

- Purpose: Canvas rendering of traces. Key functions/methods:
- – Public: render(model_snapshot)->None.
- – Internal: axis scaling, zoom label updates.
- Global/state variables: canvas handle, last bounds.
- Interactions: Receives data from controller/model; draws to UI.

EgramController.

- Purpose: Acquisition thread and scheduler. Key functions/methods:
- – Public: start()->bool, stop()->None, clear()->None, snapshot(n|None)->list[frame].
- – Internal: run loop, source adapter (mock/serial).
- Global/state variables: running flag, model/ring buffer, sampling rate.
- Interactions: Pulls frames from source; schedules UI redraw callbacks.

auth.py.

- Purpose: JSON-backed local accounts (≤10 users). Key functions/methods:
- – Public: register_user(name, pw_hash)->(bool,str), login_user(name, pw_hash)->(bool,str), logout_account()->None, count_users()->int.
- – Internal: load/save file, check duplicates/limits.
- Global/state variables: users list, data/users.json path.
- Interactions: Called by WelcomeWindow; reads/writes JSON.

3.3.3 MID — Module & Interface Description

App (main.py).

- Function: Start/stop app; route from login to dashboard.
- Purpose: To start/stop app; route from login to dashboard. Key functions/methods:
- – Public: run, _show_login, _open_dashboard
- – Internal: window lifetime control Global/state variables: root, _login_win, _dashboard.
- Interactions: On login success, destroy WelcomeWindow, create DashboardWindow; on exit, destroy all.

WelcomeWindow.

- Function: Register/login UI with dialogs and routing. Key functions/methods:
- – Public: show, on_register, on_login
- – Internal: hash pw, basic input validation Global/state variables: name/pass entries, visible.
- Interactions: on_register/on_login call auth; show info/error; on success notify App to open dashboard.

DashboardWindow.

- Purpose: Central navigation; single-instance children. Key functions/methods:
- – Public: open_help_window, open_egram, open_parameter_window
- – Internal: create-or-lift logic Global/state variables: help_window, egram_window, param_window.
- Interactions: Instantiates children on first open, then lifts; updates status labels as needed.

HelpWindow.

- Purpose: Render help topics with graceful fallback on missing files. Key functions/methods:
- – Public: update_content, load help content
- – Internal: display param document, display mode document, display text content Global/state variables: topics dict, current topic, content area.
- Interactions: Load JSON from data/…; if missing, show fallback text without crashing.

ParameterWindow.

- Purpose: Mode-based parameter editing with rounding and cross-checks. Key functions/methods:
- – Public: save, load, reset, enabled_fields
- – Internal: fill entries, disable/enable by mode Global/state variables: entries, mode var, snapshot.
- Interactions: On save, call ParamEnum setters then validate_cross; on pass, call ParameterManager.save_params and refresh entries; on load/reset, refresh immediately.

ParamEnum.

- Purpose: Enforce ranges/steps/defaults and URL > LRL. Key functions/methods:
- – Public: get_default_, set_, fields_for, validate_cross
- – Internal: is_number, round_to_step Global/state variables: DEFAULTS, RANGES/STEPS, MODES.
- Interactions: Pure computations; no GUI/file; raises errors for invalid/gap values (e.g., AA 3.3–3.4 V).

ParameterManager.

- Purpose: Robust JSON I/O for parameters with defaults on read. Key functions/methods:
- – Public: save_params, load_params, reset_params
- – Internal: ensure data dir, atomic write, default merge Global/state variables: parameters.json path, cache.
- Interactions: Save returns success/fail; load returns dict (fallback to defaults on malformed/missing).

EgramWindow.

- Purpose: User-facing controls for acquisition lifecycle and viewing. Key functions/methods:
- – Public: start_egram, stop_egram, clear_egram
- – Internal: guard double start, disable clear while running Global/state variables: running UI flags, controller/view refs.
- Interactions: Start → controller.start; if already running, ignore/warn; Stop → controller.stop; Clear only when stopped.

EgramView.

- Purpose: Draw time-series snapshots to canvas. Key functions/methods:
- – Public: render
- – Internal: scaling, zoom label Global/state variables: canvas handle, last draw.
- Interactions: Pulls snapshot from controller/model; repaints without touching controller state.

EgramController.

- Purpose: Acquire frames and schedule UI redraws safely. Key functions/methods:
- – Public: start, stop, clear, snapshot
- – Internal: acquisition loop, source adapter Global/state variables: running flag, model buffer, timers.
- Interactions: On start, spawn thread reading mock/serial; append frames; schedule UI redraw (e.g., 40–60 ms); on stop, join thread; on clear (only stopped), empty buffer.

auth.py.

- Purpose: Store and verify users with caps and error messages. Key functions/methods:
- – Public: register_user, login_user, logout_account, count_users
- – Internal: load/save JSON, duplicate/limit checks Global/state variables: users array, file path.
- Interactions: Register denies duplicates or >10; Login matches pw_hash; both return (ok, message) for UI dialogs.

3.4 Testing

3.4.1 AUTH Testing

Test ID	Purpose	Input	Output		Result
			Expected	Actual (screenshot)	
Trial1	Register new user	name=alice, pw=1234	Success dialog; user count ≤10	Figure 9	Pass
Trial2	Prevent duplicate registration	name=alice again	Error dialog: account exists	Figure 10	Pass
Trial3	Login success	valid credentials	Dashboard opens	Figure 11	Pass
Trial4	Register new user over 10	Register users over 10	Max users	Figure 12	Pass

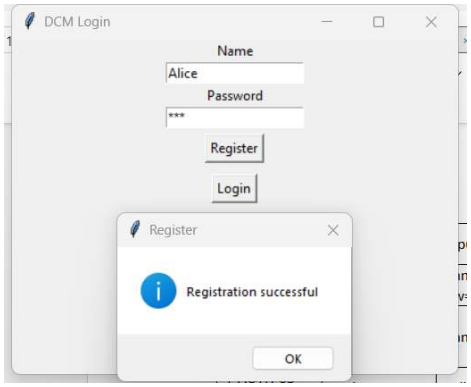


Figure 9. Trial 1: Register New User

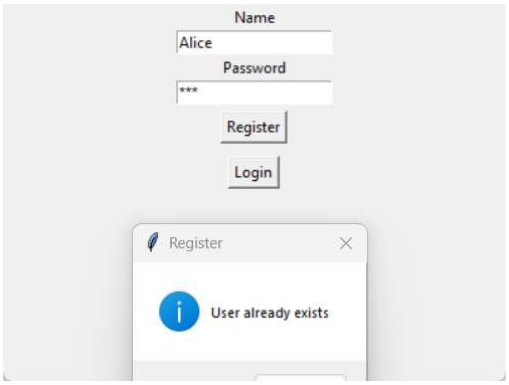


Figure 10. Trial 2: Prevent Duplicate Registration

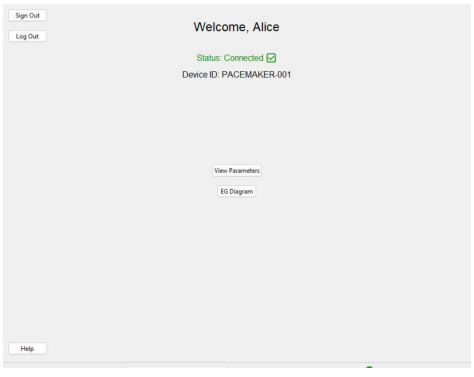


Figure 11. Trial 3: Login Success

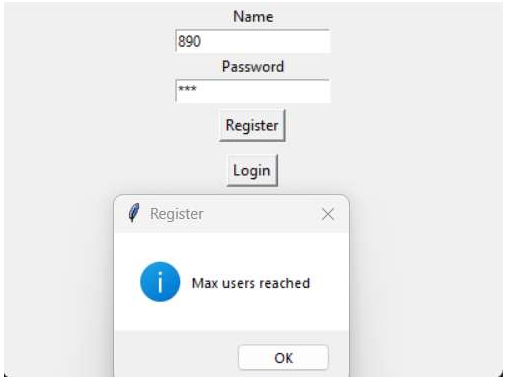


Figure 12. Trial 4: Register New User Over 10

3.4.2 MODE Testing

Test ID	Purpose	Input	Output		Result
			Expected	Actual (screenshot)	
Trial 1	Show 4 modes	Open Parameter Window → Mode dropdown	AOO/VOO/AAI/VVI visible	Figure 13	Pass

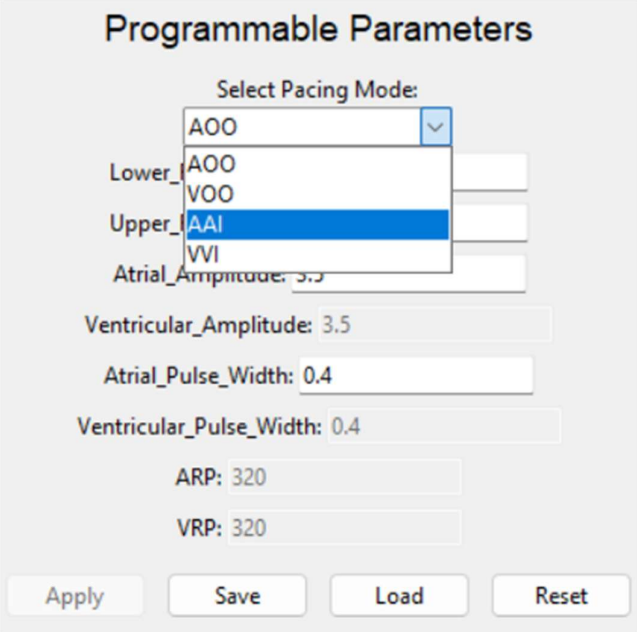


Figure 13. Trial 1: Show 4 Modes

3.4.3 PARAM Testing

Test ID	Purpose	Input	Output		Result
			Expected	Actual (screenshot)	
Trial 1	Step rounding (LRL)	LRL=57.6 → Save	Entry refreshed to 58	Figure 14	Pass
Trial 2	Cross-constraint	URL=80, LRL=90 → Save	Error dialog; not applied	Fig2	Pass
Trial 3	Invalid amplitude gap	AA=3.3 V → Save	Error dialog; not applied	Fig3	Pass
Trial 4	Reset to defaults	Click Reset	Fields reset to defaults	Fig4	Pass
Trial 5	Load refresh	Click Load	Fields immediately refresh	Fig5	Pass
Trial 6	Save success message	Set valid values → Save	Normalized values; Apply enabled	Fig6	Pass

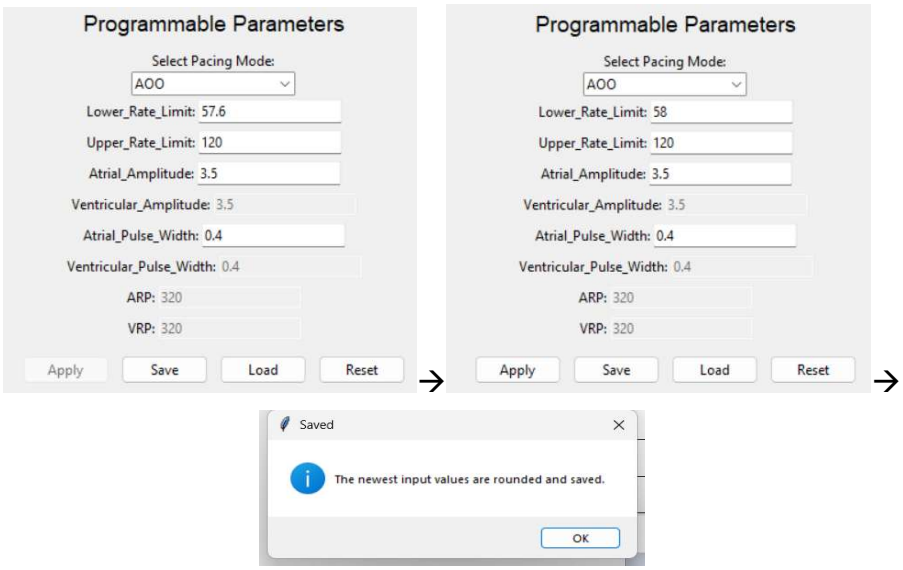


Figure 14. Trial 1: Step Rounding (LRL)

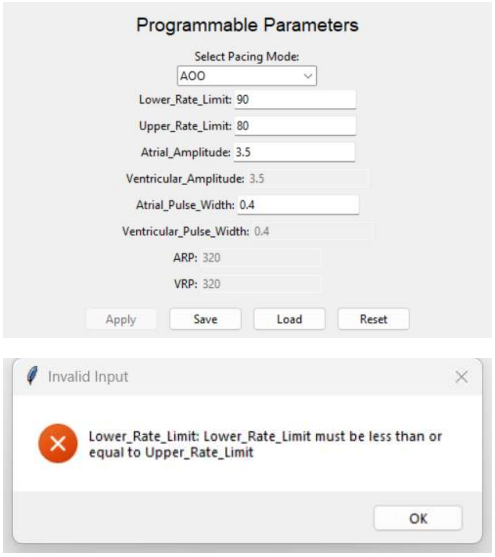


Figure 15. Trial 2: Cross-constraint

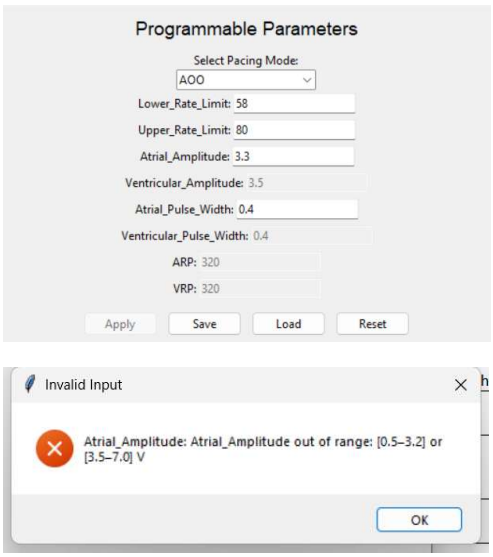


Figure 16. Trial 3: Invalid Amplitude Gap

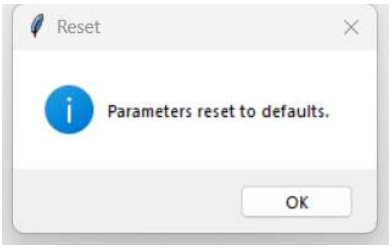


Figure 17. Trial 4: Reset to Defaults

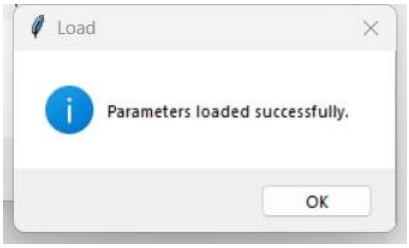


Figure 18. Trial 5: Load Refresh

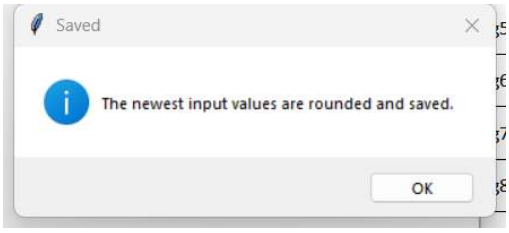


Figure 19. Trial 6: Save Success Message

3.4.4 Egram Testing

Test ID	Purpose	Input	Output		Result
			Expected	Actual (screenshot)	
Trial 1	Start plotting	Open Egram → Start	Plot updates; running flag on	Figure 20	Pass
Trial 2	Double-start guard	Click Start twice	Guard or ignore second start	Fig2	Pass

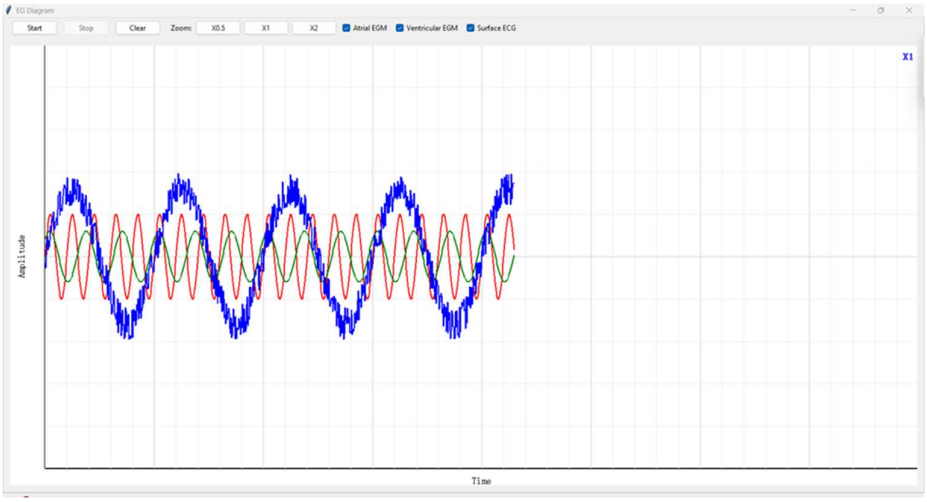


Figure 20. Trial 1: Start Plotting

Figure 21. Trial 2: Double-Start Guard

3.4.5 Help Testing

Test ID	Purpose	Input	Output		Result
			Expected	Actual (screenshot)	
Trial 1	Missing docs fallback	Remove a JSON doc	Fallback tip shown; no crash	Fig1	Pass
Trial 2	Self_connect is false	Change self_connect to false	Status disconnected	Fig2	Pass
Trial 3	Self out of range is true	Change self out of range to true	Communication out of range	Fig3	Pass
Trial 4	Self_noise unstable is true	Change self noise unstable to true	Noise/unstable serial connection	Fig4	Pass

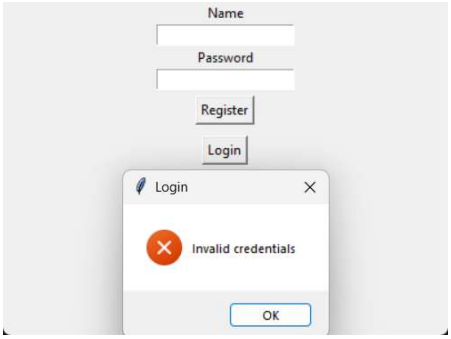


Figure 22. Trial 1: Missing Docs Fallback

Figure 23. Trial 2: Self_connect is false

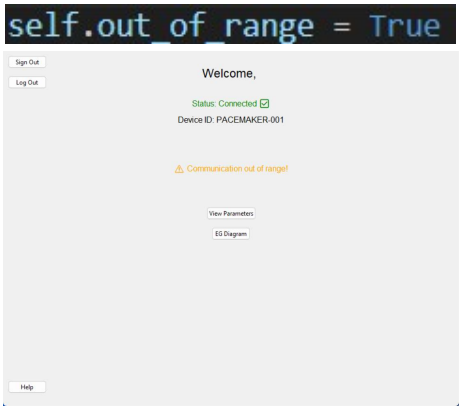


Figure 24. Trial 3: Self out of range is true

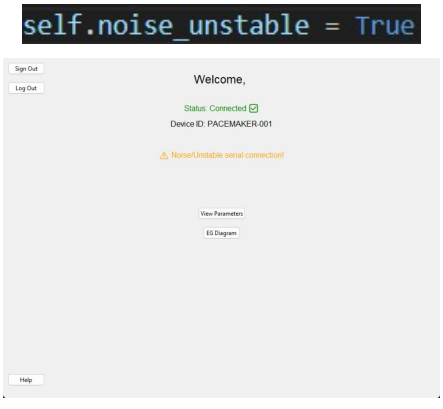


Figure 25. Trial 4: Self_noise unstable is true

3.4.6 AOO Testing

Test 1

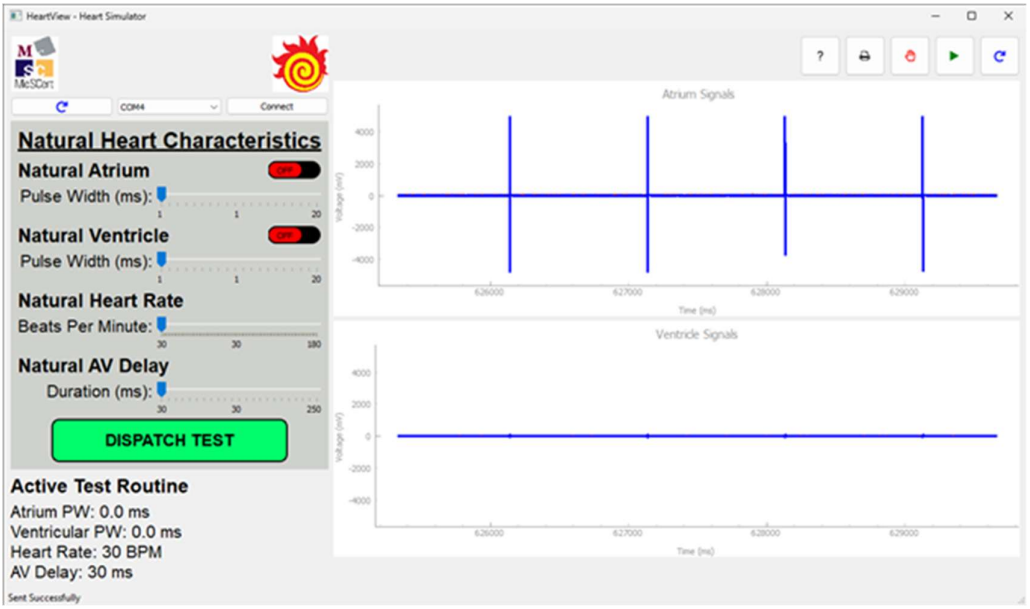
Purpose: Verify AOO delivers atrial pacing with specified parameters.

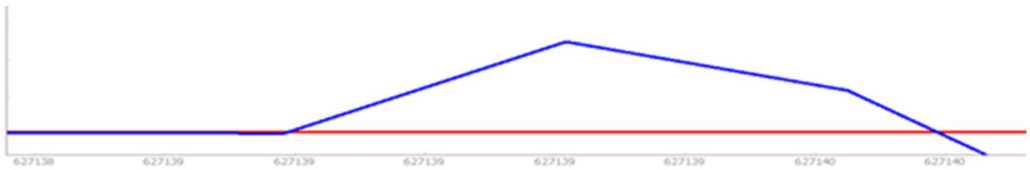
Input conditions:

- Amp: 5
- LRL: 60
- Pulse Width: 1
- Mode: 0

Expected output: Atrium signals every 1000ms with an amplitude of 5V. A pulse width of 1ms.

Actual output:





Result: Pass

Test 2

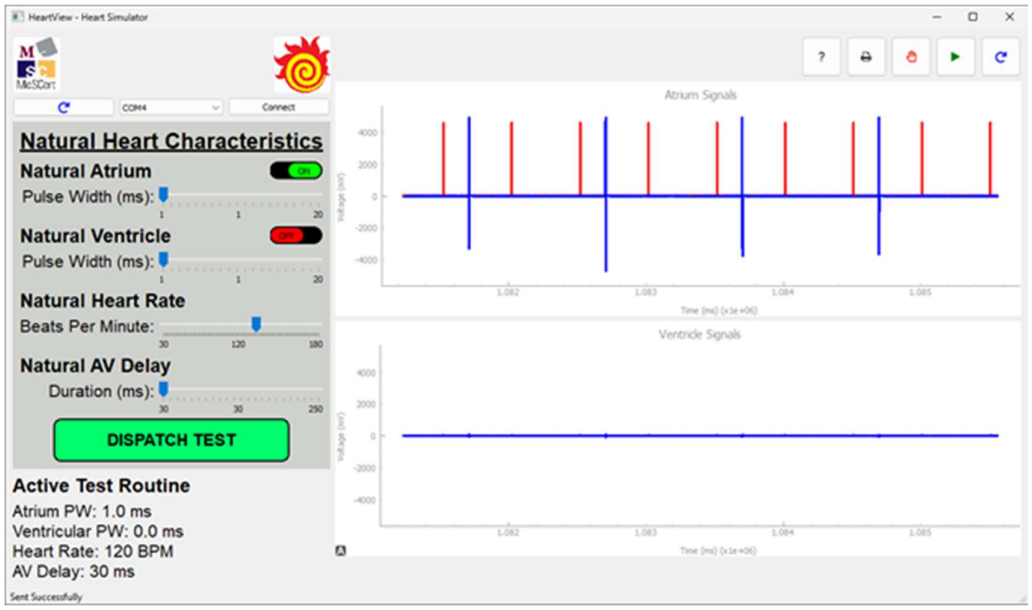
Purpose: Verify AOO continues pacing independently and ignores intrinsic atrial activity.

Input conditions:

- Amp: 5
- LRL: 60
- Pulse Width: 1
- Mode: 0
- Heartview Natural Heart Rate: 120 bpm (faster than LRL to simulate intrinsic activity)

Expected output: Both paced and natural atrial events are visible.

Actual output:



Result: Pass

Test 3

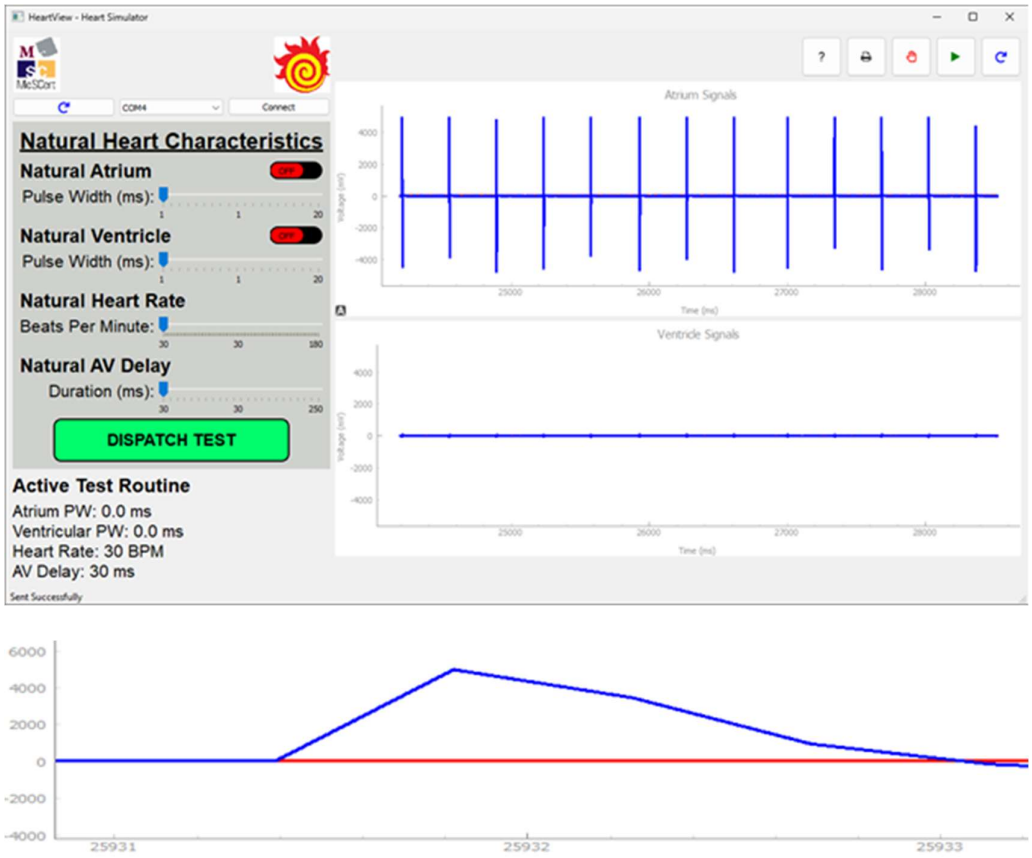
Purpose: Verify parameter validation clamps invalid inputs to safe specification limits.

Input conditions:

- Amp: 100 (invalid - spec max is 5)
- LRL: 1000 (invalid - spec max is 175)
- Pulse Width: 100 (invalid - spec max is 1.9)
- Mode: 0

Expected output: System rejects or clamps invalid parameters to specification limits.

Actual output:



Result: Pass

3.4.7 VOO Testing

Test 1:

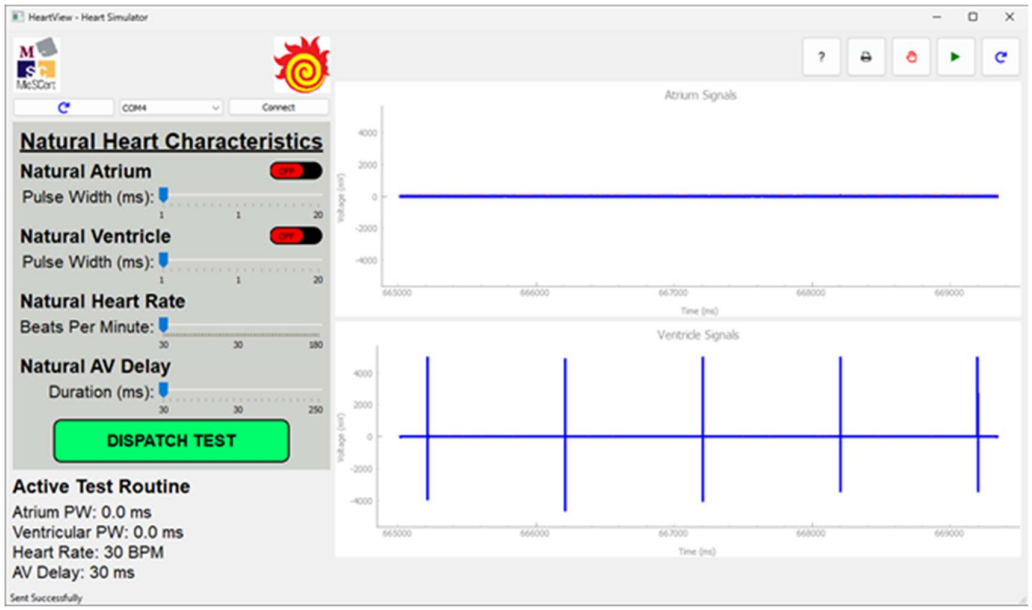
Purpose of the test: Verify VOO delivers ventricular pacing with specified parameters.

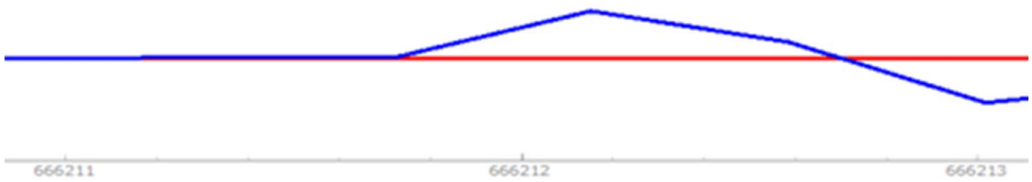
Input conditions:

- Amp: 5
- LRL: 60
- Pulse Width: 1
- Mode: 1

Expected output: Ventricle signals every 1000ms with an amplitude of 5V and a pulse width of 1ms.

Actual output:





Result: Pass

Test 2

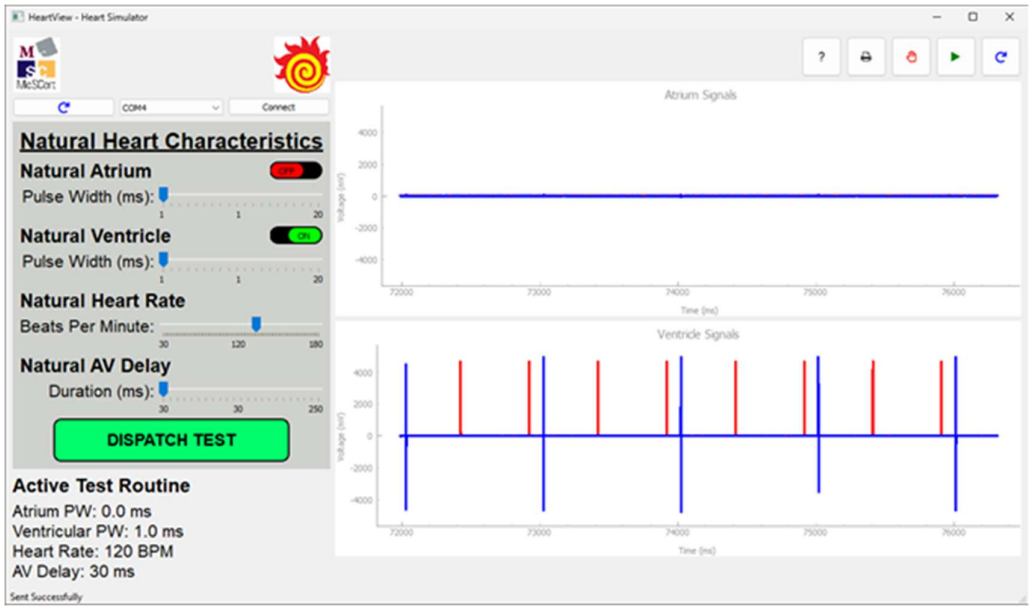
Purpose: Verify VOO continues pacing independently and ignores intrinsic ventricular activity.

Input conditions:

- Amp: 5
- LRL: 60
- Pulse Width: 1
- Mode: 1
- Heartview Natural Heart Rate: 120 bpm (faster than LRL to simulate intrinsic activity)

Expected output: Both paced and natural ventricular events are visible.

Actual output:



Result: Pass

Test 3

Purpose: Verify parameter validation clamps invalid inputs to safe specification limits.

Input conditions:

- Amp: 100 (invalid - spec max is 5)
- LRL: 1000 (invalid - spec max is 175)
- Pulse Width: 100 (invalid - spec max is 1.9)
- Mode: 1

Expected output: System rejects or clamps invalid parameters to specification limits.

Actual output:

Result: Pass

3.4.8 AAI Testing

Test 1

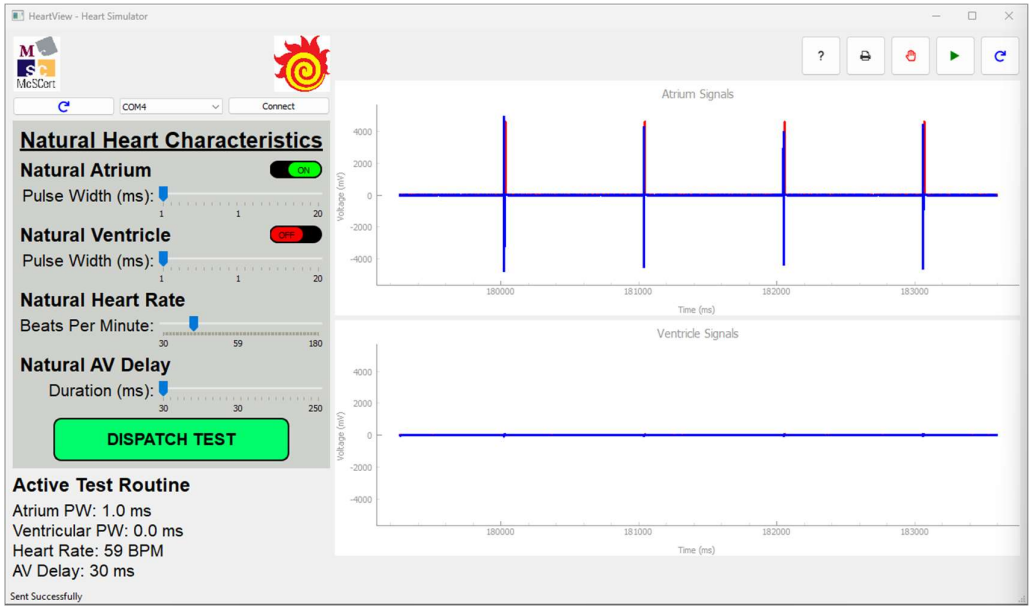
Purpose: Verify AAI mode delivers atrial pacing when intrinsic atrial activity falls below LRL.

Input conditions:

- Amplitude: 5
- Sensitivity: 8
- LRL: 60
- Pulse Width: 1
- ARP: 200
- Mode: 2
- Natural atrial rate: 59

Expected output: Pacing pulse occurs.

Actual output:



(Intrinsic atrial pulse is difficult to see due to overlapping with the pacing pulse)

Result: Pass

Test 2

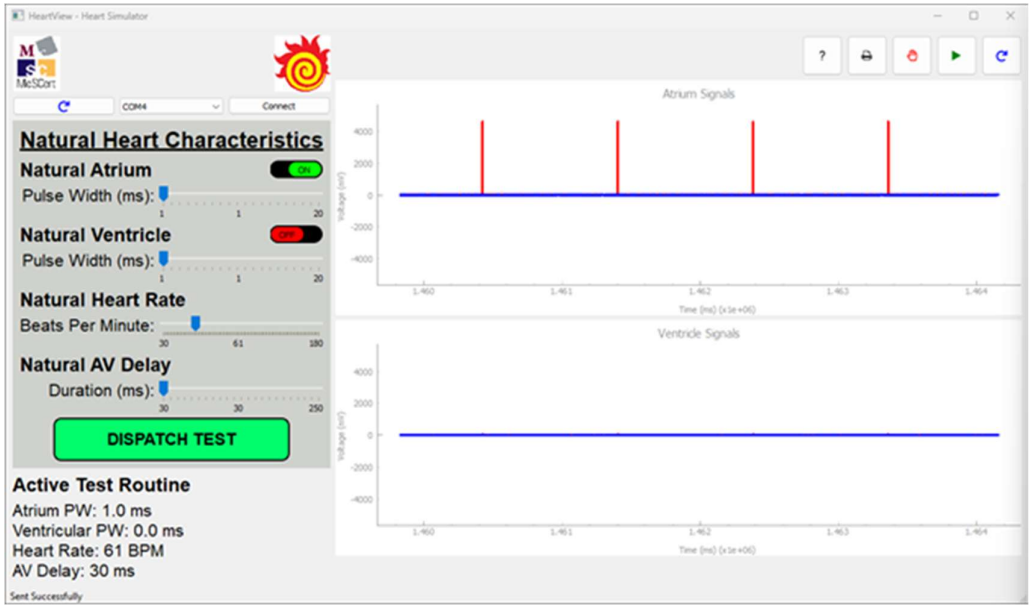
Purpose: Verify AAI mode does not deliver atrial pacing when intrinsic atrial activity exceeds LRL.

Input conditions:

- Amplitude: 5
- Sensitivity: 8
- LRL: 60
- Pulse Width: 1
- ARP: 200
- Mode: 2
- Natural atrial rate: 61

Expected output: Pacing pulse does not occur.

Actual output:



Result: Pass

Test 3

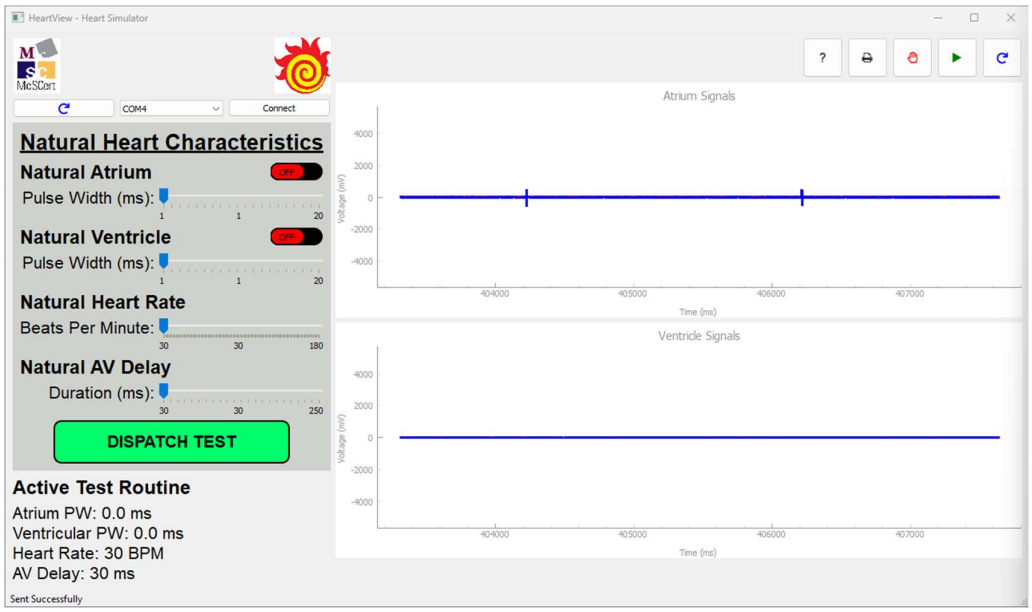
Purpose: Verify AAI mode delivers pacing at the Lower Rate Limit when no natural atrial activity is detected and validate parameter clamping to specification limits.

Input conditions:

- Amplitude: 0 (invalid - spec min is 0.5V)
- Sensitivity: 8
- LRL: 0 (invalid - spec min is 30)
- Pulse Width: 1
- ARP: 200
- Mode: 2

Expected output: System clamps invalid parameters to specification minimums. Atrium pacing occurs every 2000ms with a 0.5V amplitude.

Actual output:



Result: Pass

3.4.9 VVI Testing

Test 1

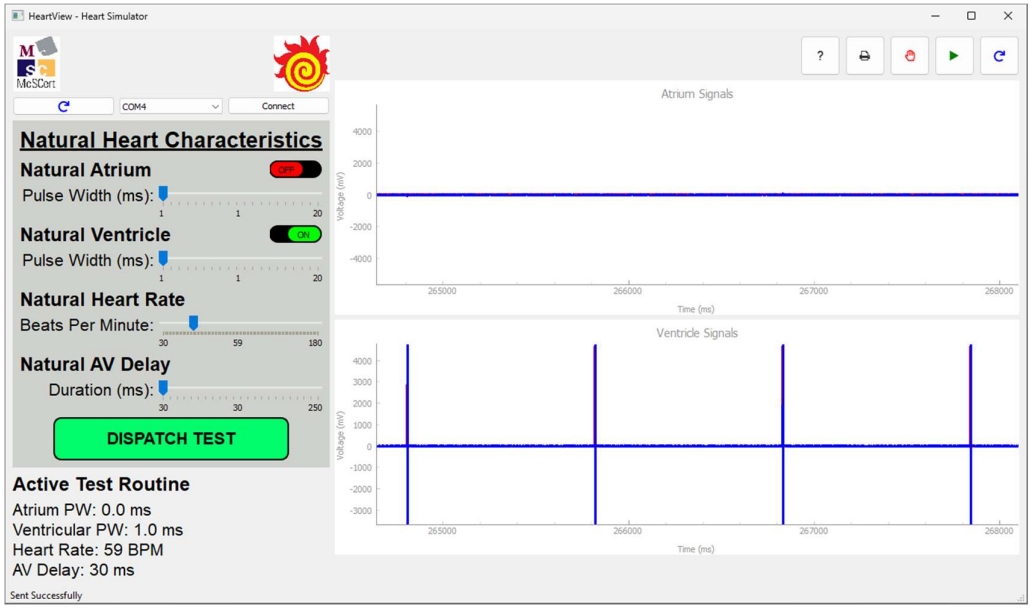
Purpose: Verify VVI mode delivers ventricular pacing when intrinsic ventricular activity falls below LRL.

Input conditions:

- Amplitude: 5
- Sensitivity: 8
- LRL: 60
- Pulse Width: 1
- VRP: 200
- Mode: 3
- Natural ventricular rate: 59

Expected output: Pacing pulse occurs.

Actual output:



(Intrinsic ventricular pulse is difficult to see due to overlapping with the pacing pulse)

Result: Pass

Test 2

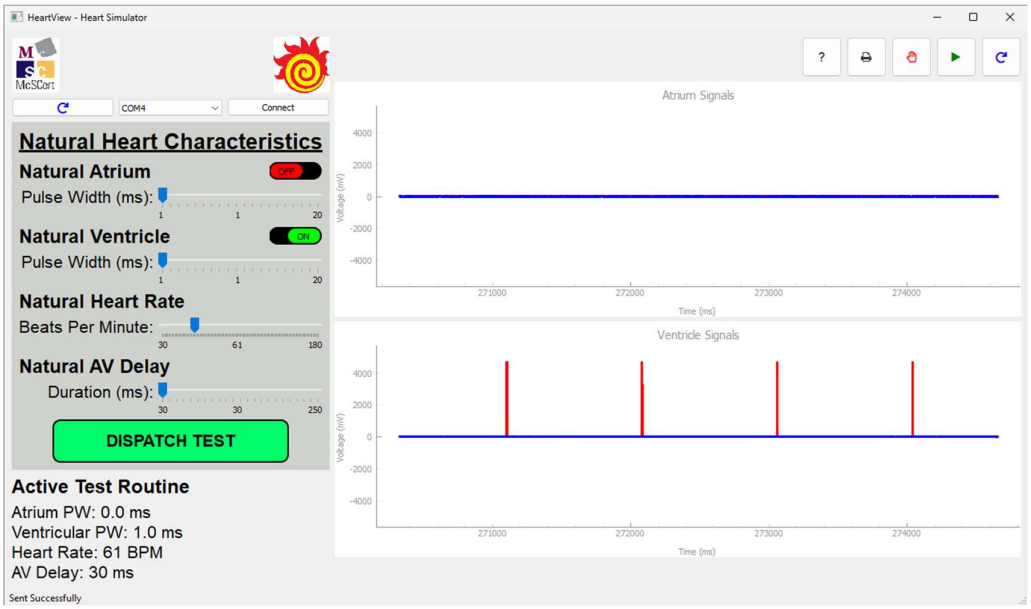
Purpose: Verify VVI mode does not deliver ventricular pacing when intrinsic ventricular activity exceeds LRL.

Input conditions:

- Amplitude: 5
- Sensitivity: 8
- LRL: 60
- Pulse Width: 1
- VRP: 200
- Mode: 3
- Natural atrial rate: 61

Expected output: Pacing pulse does not occur.

Actual output:



Result: Pass

Test 3

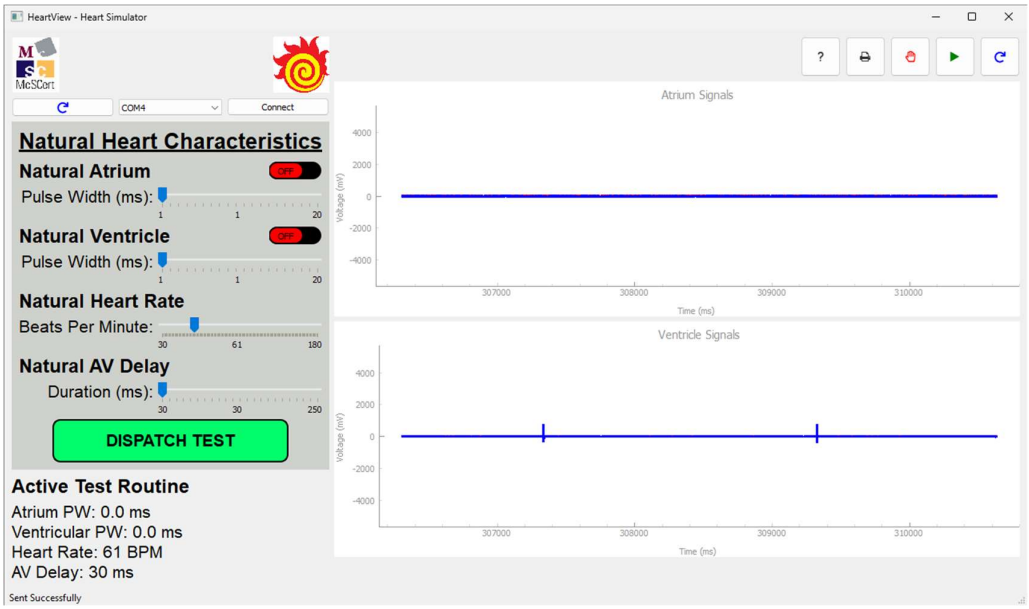
Purpose: Verify VVI mode delivers pacing at the Lower Rate Limit when no natural ventricular activity is detected and validate parameter clamping to specification limits.

Input conditions:

- Amplitude: 0 (invalid - spec min is 0.5V)
- Sensitivity: 8
- LRL: 0 (invalid - spec min is 30)
- Pulse Width: 1
- VRP: 200
- Mode: 3

Expected output: System clamps invalid parameters to specification minimums. ventricle pacing occurs every 2000 ms with a 0.5 V amplitude.

Actual output:



Result: Pass

3.5 GenAI Usage

In this assignment, generative AI was primarily used for grammar polishing and refinement of the report (note: the report was not directly generated by AI; one of our team members is not a native English speaker, so AI was used for translation and polishing during the writing process). In the DCM UI design, AI was employed to check for identified issues (such as abnormal repeated window creation and destruction problems) and was used only as a reference for bug fixes.