Deliverable 2

MECHTRON 3K04

Group #20

Nov 27, 2025

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 is intended to present a secure, consistent, and user-friendly interface across the Pacemaker for physicians to configure, store, transmit, and validate all programmable pacemaker settings. Doctors select pacing mode, rate limits, refractory periods, pulse features, and rate-adaptive settings from the DCM, while all values entered are validated, step-rounded, and cross-checked for safety prior to application (For both D1 and D2).

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, VVI, AOOR, VOOR, AAIR, and VVIR modes with parameters specified in the Deliverables 1 & 2 document.

DCM: Develop an interface that enables user registration and login, displays pacing modes, and allows input and storage of modifiable parameters. For deliverable 2, DCM can do the following. Firstly, establish a secure serial connection to the pacemaker device. Secondly, upload clinician-configured parameters within a structured communication frame, then, verify that the parameters stored on the device match those entered in the DCM, and receive and display real-time atrial and for monitoring and validation.

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.

Deliverable 2 is to extend and complete the pacemaker system by:

- Implementing advanced rate-adaptive pacing modes: AOOR, VOOR, AAIR, VVIR
- Establishing safety assurance mechanisms
- Integrating bidirectional serial communication with the DCM
- Providing comprehensive testing and validation of the system design.
- Providing a serial communication interface for connecting, disconnecting, and identifying pacemaker devices.
- Generating and transmitting a structured parameter frame that encodes all programmable values.
- Verifying that parameters stored on the pacemaker match the clinician’s input.
- Updating the UI to reflect connection state, device identity, and communication quality.

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
AOOR	Atrium	Atrium	Inhibited	Paces the atrium only if no intrinsic atrial activity is sensed within the programmed interval. Rate adapts to detected physical activity via accelerometer. Pacing rate increases from LRL toward MSR based on activity level, reaction time, and response factor.
VOOR	Ventricle	Ventricle	Inhibited	Paces the ventricle only if no intrinsic ventricular activity is sensed within the programmed interval. Rate adapts to detected physical activity via accelerometer. Pacing rate increases from LRL toward MSR based on activity level, reaction time, and response

				factor.

AAIR	Atrium	Atrium	Inhibited	Paces the atrium only if no intrinsic atrial activity is sensed within the programmed interval. Rate adapts to detected physical activity via accelerometer. Pacing rate increases from LRL toward MSR based on activity level, reaction time, and response factor.
VVIR	Ventricle	Ventricle	Inhibited	Paces the ventricle only if no intrinsic ventricular activity is sensed within the programmed interval. Rate adapts to detected physical activity via accelerometer. Pacing rate increases from LRL toward MSR based on activity level, reaction time, and response factor.

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 for deliverable 2 extends the d1 architecture to support all eight-pacing modes with integrated rate adaptation. The core three-module structure (Input Hardware Hiding, Main, and Output Hardware Hiding) is retained and enhanced by incorporating rate adaptation into the module that processes accelerometer data and adjusts the pacing intervals dynamically for the R-modes. In addition, one new module will be added: COM represents the input and output of the serial port.

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.

For Deliverable 2, the Input Hardware Hiding module has been enhanced to incorporate rate adaptation logic. The rate adaptation block processes real-time accelerometer data from the on-board sensor to dynamically adjust the Lower Rate Limit (LRL) based on detected activity levels. When sufficient activity is detected through accelerometer measurements, the module increases the LRL to enable rate-responsive pacing modes (AOOR, VOOR, AAIR, VVIR). The rate adaptation logic applies delays and configurable thresholds to determine activity levels and transitions the LRL smoothly to prevent sudden changes in pacing rate.

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.

For Deliverable 2, the Main module has been expanded to include rate-adaptive functionality in the four new rate-responsive modes (AOOR, VOOR, AAIR, VVIR). In these modes, the Lower Rate Limit (LRL) is no longer fixed but is dynamically adjusted based on accelerometer data processed by the Input Hardware Hiding module.

Output Hardware Hiding

The Output Hardware Hiding module takes in the outputs of the Main module as inputs and converts 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.

For Deliverable 2, the Output Hardware Hiding module has been enhanced to support egram signal acquisition and output in addition to the existing pacing and sensing control signals. The module now routes atrial and ventricular egram signal from the sensing circuitry to the DCM over the serial communication interface for real-time display and analysis. Additionally, the module has been updated to handle the expanded set of control outputs required by the four new rate-adaptive modes: AOOR, VOOR, AAIR, VVIR, ensuring that dynamic rate adjustments computed in the Main module are properly translated to PWM control signals.

COM

The COM Module serves as the bidirectional interface between the pacemaker device and the Device Controller-Monitor (DCM). This module manages serial communication protocols, data formatting, and signal conditioning for all data exchanged between the hardware and the DCM application. The COM_IN input channel receives programmable parameters and commands from the DCM over the serial communication link. It then outputs these parameters in a format compatible with the input hardware hiding module. The COM_OUT output channel transmits real-time pacemaker status, diagnostic data, and egram signals from the device to the DCM for display and analysis.

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
Maximum Sensor Rate (MSR)	The upper limit on pacing rate that the pacemaker will achieve in response to detected patient activity, preventing excessively rapid pacing.	50 ~ 175	BPM
Activity Threshold	The accelerometer signal level above which the pacemaker recognizes patient physical activity and begins increasing the pacing rate.	0 ~ 5	-

Response Factor	A programmable gain parameter that controls how aggressively the pacing rate increases relative to detected activity levels.	1 ~ 16	-
Reaction Time	The time delay governing how quickly the pacemaker accelerates from baseline LRL toward the sensor-indicated rate when activity is first detected.	10 ~ 50	sec
Recovery Time	The time interval over which the pacemaker decelerates from the elevated sensor-indicated rate back to baseline LRL after activity subsides.	2 ~ 16	min
Adapted Rate	The dynamically computed pacing interval resulting from applying the rate adaptation logic in response to current accelerometer activity, constrained between LRL and MSR.	333 ~ 2000	ms

Mode	The operational pacing mode of the device.
	AOO = 0, VOO = 1, AAI = 2, VVI = 3
	AOOR=4, VOOR=5, AAIR=6, VVIR=7
		0, 1, 2, 3,4,5,6,7	-

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
MAG_ACCEL	N/A	The magnitude of the data from the three-axis on-board accelerometer used to detect patient physical activity for the four rate-adaptive modes	Device dependent

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 eight implemented modes (AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, VVIR) 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

AOOR

AOOR mode paces the atrium at a rate that adapts to patient activity detected by the accelerometer. The pacing rate increases from the baseline Lower Rate Limit (LRL) toward the Maximum Sensor Rate (MSR) as activity increases and decreases back to LRL when activity stops.

Current State	Trigger	Actions	Next State
AOOR_Entry	After completing all actions	Set pacing voltage Turn off all ventricle controls, initialize rate adaption	AOOR_Charging
AOOR_Charging	After (adapted_rate_ms - pulse_width_ms) time in ms	Stop discharging capacitor to atrium Charge Capacitor	AOOR_Pacing
AOOR_Pacing	After pulse_width time in ms	Stop charging capacitor Discharge capacitor to atrium	AOOR_Charging

VOOR

VOOR mode paces the ventricle at a rate that adapts to patient activity. The ventricular pacing rate increases and decreases based on activity detected by the built-in accelerometer.

Current State	Trigger	Actions	Next State
VOOR_Entry	After completing all actions	Set pacing voltage Turn off all ventricle controls, initialize rate adaption	VOOR_Charging
VOOR_Charging	After (adapted_rate_ms - pulse_width_ms) time in ms	Stop discharging capacitor to ventricle, Charge Capacitor	VOOR_Pacing
VOOR_Pacing	After pulse_width time in ms	Stop charging capacitor, Discharge capacitor to atrium	VOOR_Charging

AAIR

AAIR mode senses intrinsic atrial activity and starts pacing if a natural beat is detected within the set adapted pacing range. If no intrinsic activity is sensed, the pacemaker delivers an atrial pace at the adapted rate that is adjusted based on the activity detected by the accelerometer.

Current State	Trigger	Actions	Next State
AAIR_Entry	After completing all actions	Set pacing voltage Turn off all ventricle controls, Stop discharging capacitor to atrium, Charge Capacitor, Initialize rate adaption	AAIR_Idle
AAIR_Idle	Case 1: Heartbeat detected in atrium	Turn off Red LED, set heartbeat_detected flag	AAIR_Detected
	Case 2: Heartbeat not detected after adapted_rate_ms – pulse_width – heartbeat_detected*arp time	Proceed to pacing	AAIR_Pacing
AAIR_Detected	After ARP time in ms	Set local variable “heartbeat_detected” to 1	AAIR_Idle
AAIR_Pacing	After (pulse_width) time in ms	Stop charging capacitor, Discharge capacitor to atrium, Turn on Red LED	AAIR_Charging
AAIR_Charging	After completing all actions	Stop discharging capacitor to atrium, Charge Capacitor	AAIR_Idle

VVIR

VVIR mode senses intrinsic ventricular activity and starts pacing if a natural beat is detected within the set adapted pacing range. If no intrinsic activity is sensed, the pacemaker delivers a ventricular pace at the adapted rate that is adjusted based on the activity detected by the accelerometer.

Current State	Trigger	Actions	Next State
VVIR_Entry	After completing all actions	Set pacing voltage Turn off all atrium controls Stop discharging capacitor to ventricle Charge Capacitor, Initialize rate adaption	VVIR_Idle
VVIR_Idle	Case 1: Heartbeat detected in ventricle	Turn off Blue LED, set heartbeat_detected flag	VVIR_Detected
	Case 2: Heartbeat not detected after adapted_rate_ms – pulse_width – heartbeat_detected*arp time	Proceed to pacing	VVIR_Pacing
VVIR_Detected	After VRP time in ms	Set local variable “heartbeat_detected” to 1	VVIR_Idle
VVIR_Pacing	After (pulse_width) time in ms	Stop charging capacitor, Discharge capacitor to ventricle, Turn on Blue LED	VVIR_Charging
VVIR_Charging	After completing all actions	Stop discharging capacitor to ventricle,	VVIR_Idle

		Charge Capacitor	

2.3.5 Simulink diagram

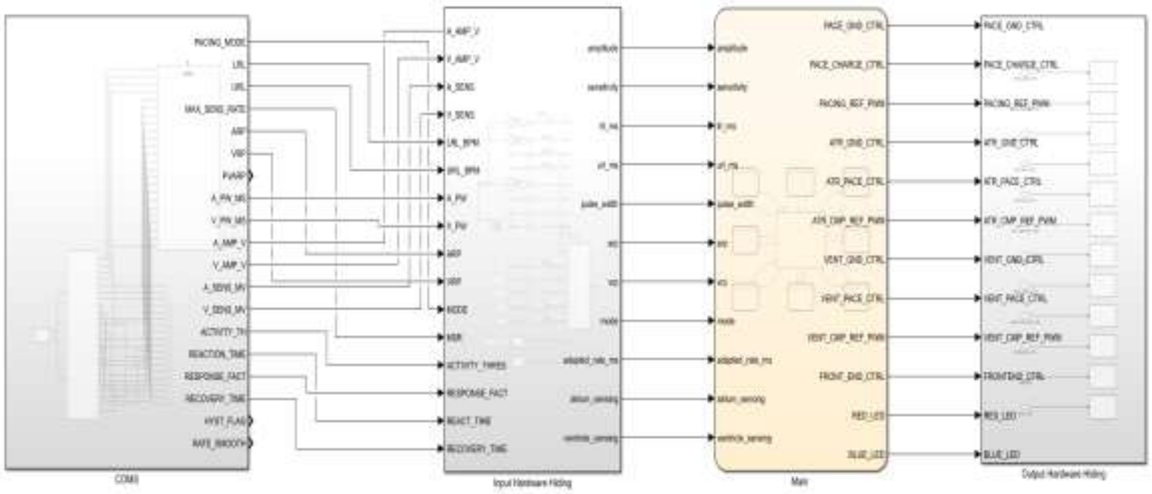


Figure 1. Full Simulink Model

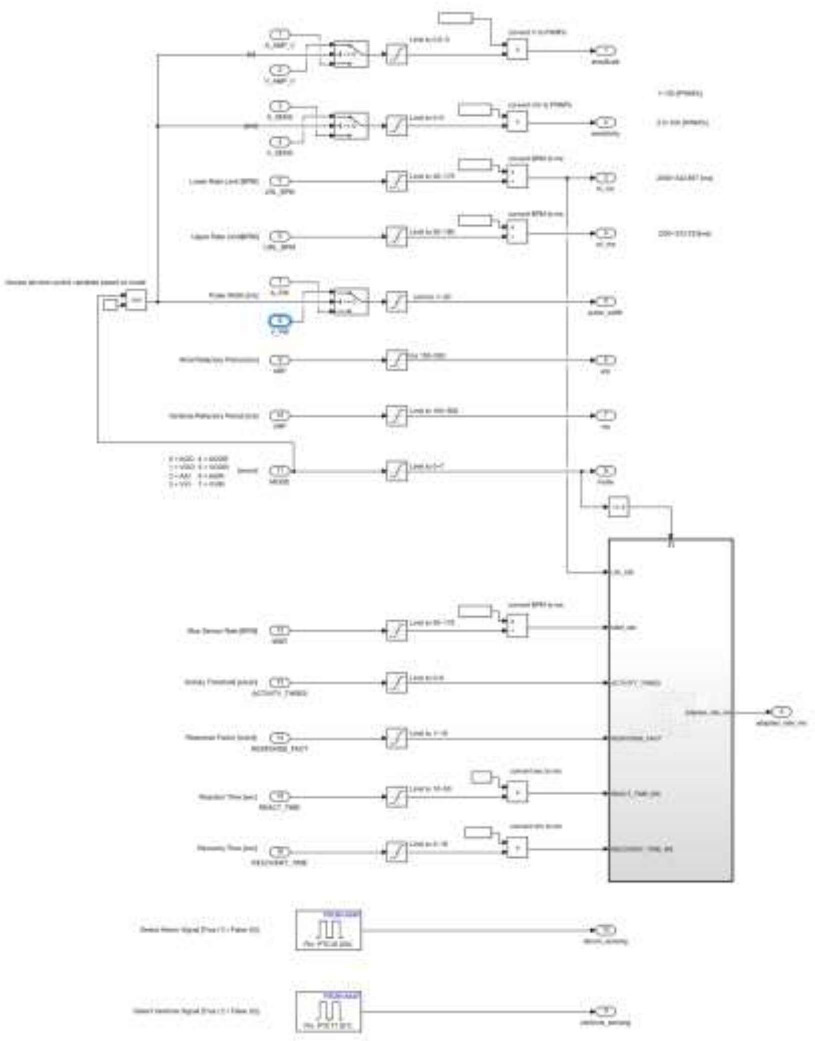


Figure 2. Input Hardware Hiding Module

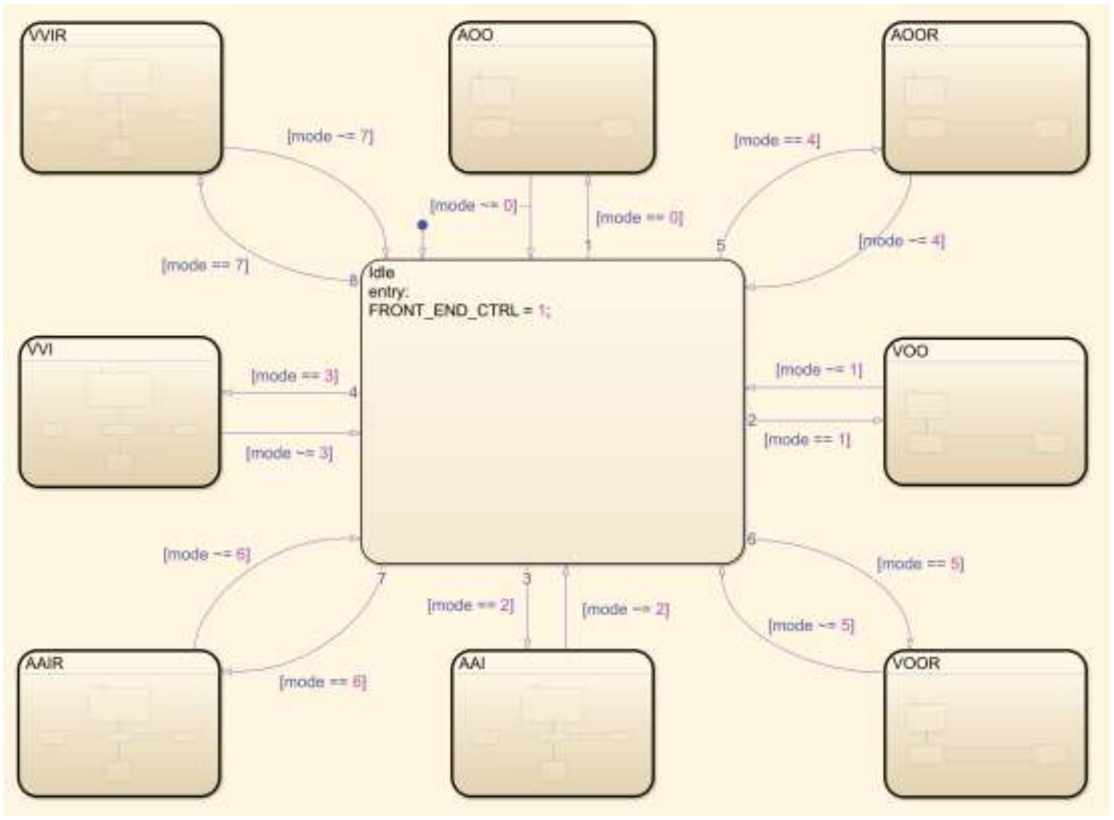
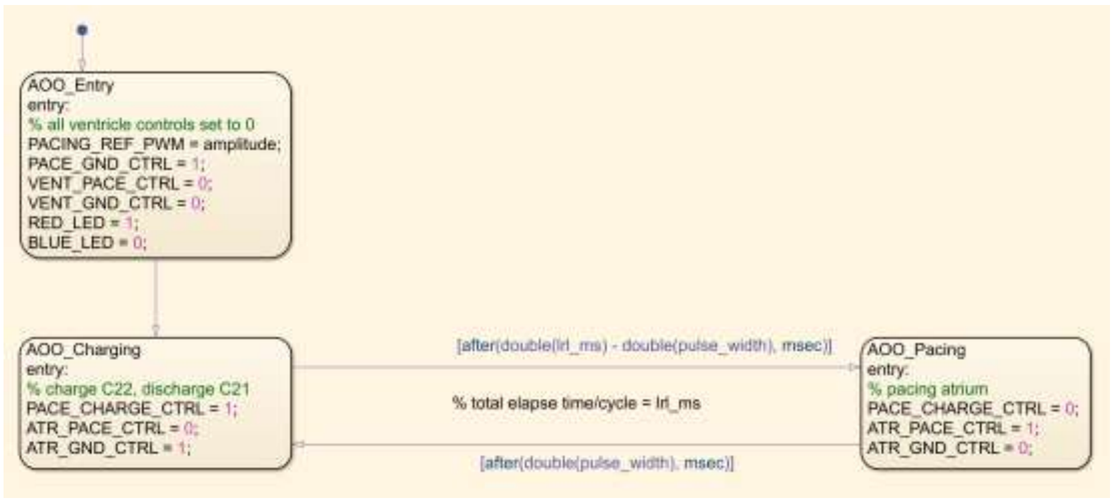


Figure 3. Main Module



```
AOO_Entry
entry:
% all ventricle controls set to 0
PACING_REF_PWM = amplitude;
PACE_GND_CTRL = 1;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 0;
RED_LED = 1;
BLUE_LED = 0;
```

```
AOO_Charging
entry:
% charge C22, discharge C21
PACE_CHARGE_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 1;
```

```
[after(double(lrl_ms) - double(pulse_width), msec)]
% total elapse time/cycle = lrl_ms
[after(double(pulse_width), msec)]
```

```
AOO_Pacing
entry:
% pacing atrium
PACE_CHARGE_CTRL = 0;
ATR_PACE_CTRL = 1;
ATR_GND_CTRL = 0;
```

Figure 4. AOO Submodule in Main

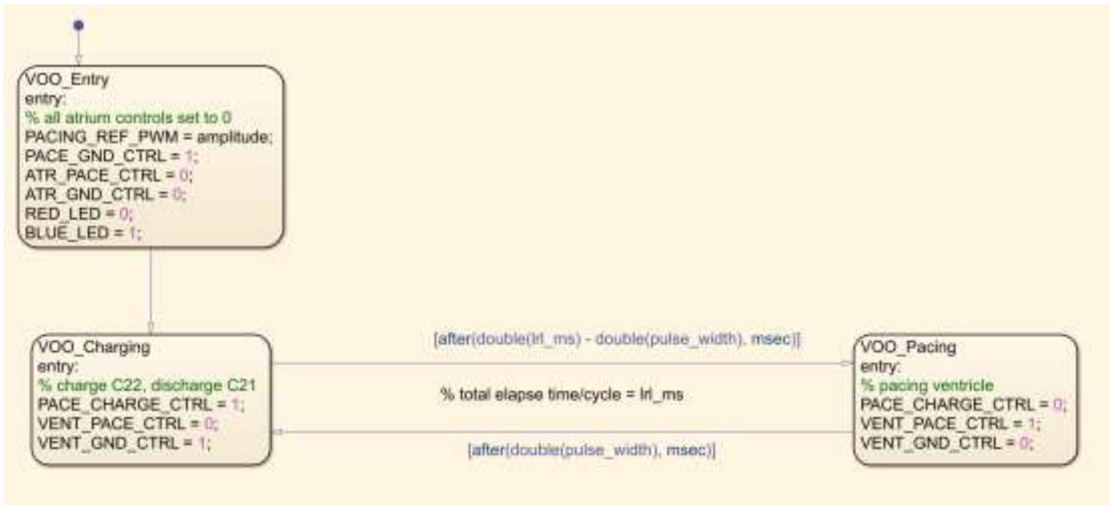


Figure 5. VOO Submodule in Main

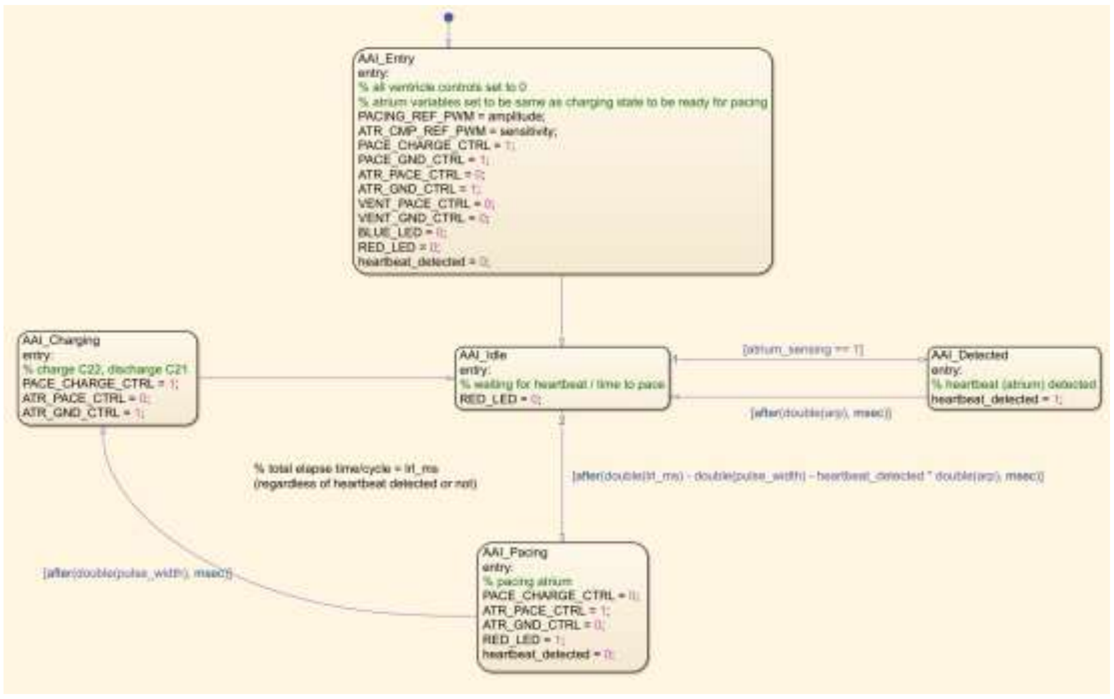
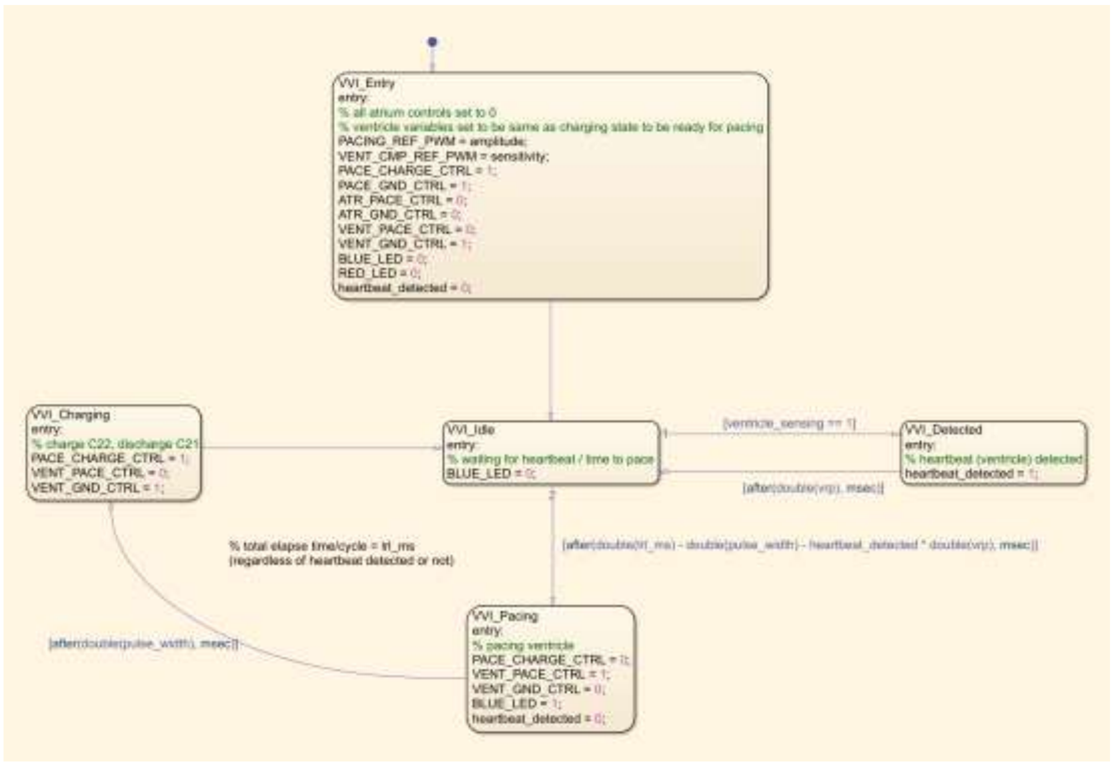

AAI_Entry
entry:
% all ventricle controls set to 0
% atrium variables set to be same as charging state to be ready for pacing
PACING_REF_PWM = amplitude;
ATR_CMP_REF_PWM = sensitivity;
PACE_CHARGE_CTRL = 1;
PACE_GND_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 1;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 0;
BLUE_LED = 0;
RED_LED = 0;
heartbeat_detected = 0;
AAI_Charging
entry:
% charge C22, discharge C21
PACE_CHARGE_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 1;
AAI_Idle
entry:
% waiting for heartbeat / time to pace
RED_LED = 0;
[atrium_sensing == 1]
AAI_Detected
entry:
% heartbeat (atrium) detected
heartbeat_detected = 1;
[after(double(arp), msec)]
% total elapse time/cycle = lrl_ms
(regardless of heartbeat detected or not)
[after(double(lrl_ms) - double(pulse_width) - heartbeat_detected * double(arp), msec)]
[after(double(pulse_width), msec)]
AAI_Pacing
entry:
% pacing atrium
PACE_CHARGE_CTRL = 0;
ATR_PACE_CTRL = 1;
ATR_GND_CTRL = 0;
RED_LED = 1;
heartbeat_detected = 0;

Figure 6. AAI Submodule in Main



```
VVI_Entry
entry:
% all atrium controls set to 0
% ventricle variables set to be same as charging state to be ready for pacing
PACING_REF_PWM = amplitude;
VENT_CMP_REF_PWM = sensitivity;
PACE_CHARGE_CTRL = 1;
PACE_GND_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 0;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 1;
BLUE_LED = 0;
RED_LED = 0;
heartbeat_detected = 0;
```

```
VVI_Charging
entry:
% charge C22, discharge C21
PACE_CHARGE_CTRL = 1;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 1;
```

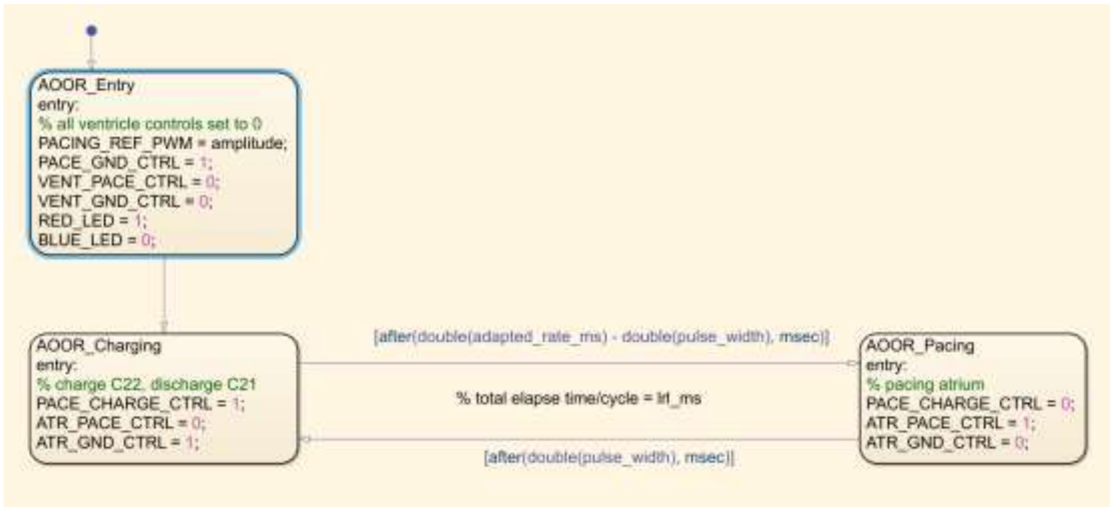
```
VVI_Idle
entry:
% waiting for heartbeat / time to pace
BLUE_LED = 0;
```

```
VVI_Detected
entry:
% heartbeat (ventricle) detected
heartbeat_detected = 1;
```

```
VVI_Pacing
entry:
% pacing ventricle
PACE_CHARGE_CTRL = 1;
VENT_PACE_CTRL = 1;
VENT_GND_CTRL = 0;
BLUE_LED = 1;
heartbeat_detected = 0;
```

[ventricle_sensing == 1]
[after(double(vrp), msec)]
% total elapse time/cycle = lrl_ms
(regardless of heartbeat detected or not)
[after(double(lrl_ms) - double(pulse_width) - heartbeat_detected * double(vrp), msec)]
[after(double(pulse_width), msec)]

Figure 7. VVI Submodule in Main



```
AOOR_Entry
entry:
% all ventricle controls set to 0
PACING_REF_PWM = amplitude;
PACE_GND_CTRL = 1;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 0;
RED_LED = 1;
BLUE_LED = 0;
```

```
AOOR_Charging
entry:
% charge C22, discharge C21
PACE_CHARGE_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 1;
```

```
AOOR_Pacing
entry:
% pacing atrium
PACE_CHARGE_CTRL = 0;
ATR_PACE_CTRL = 1;
ATR_GND_CTRL = 0;
```

[after(double(adapted_rate_ms) - double(pulse_width), msec)]
% total elapse time/cycle = lrl_ms
[after(double(pulse_width), msec)]

Figure 8. AOOR Submodule in Main

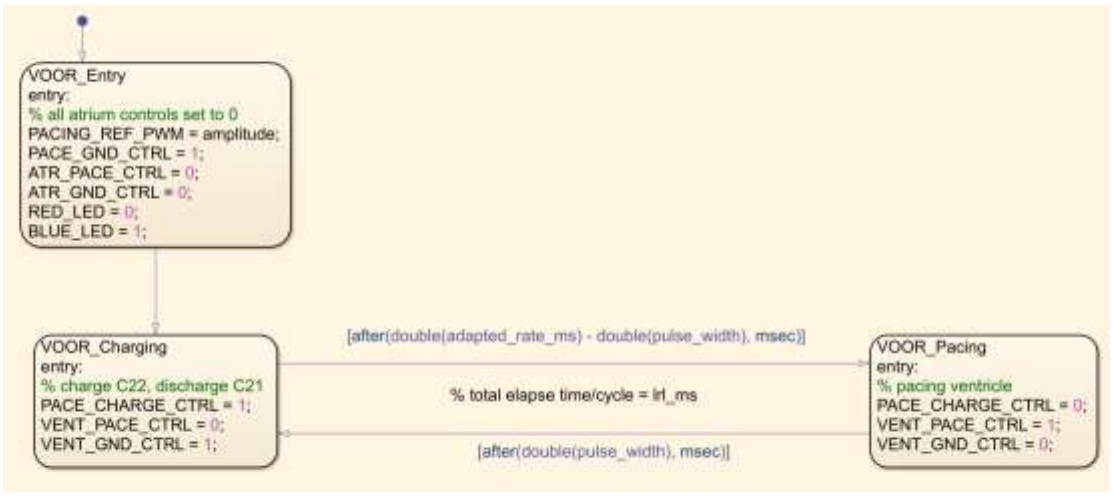


Figure 9. VOOR Submodule in Main

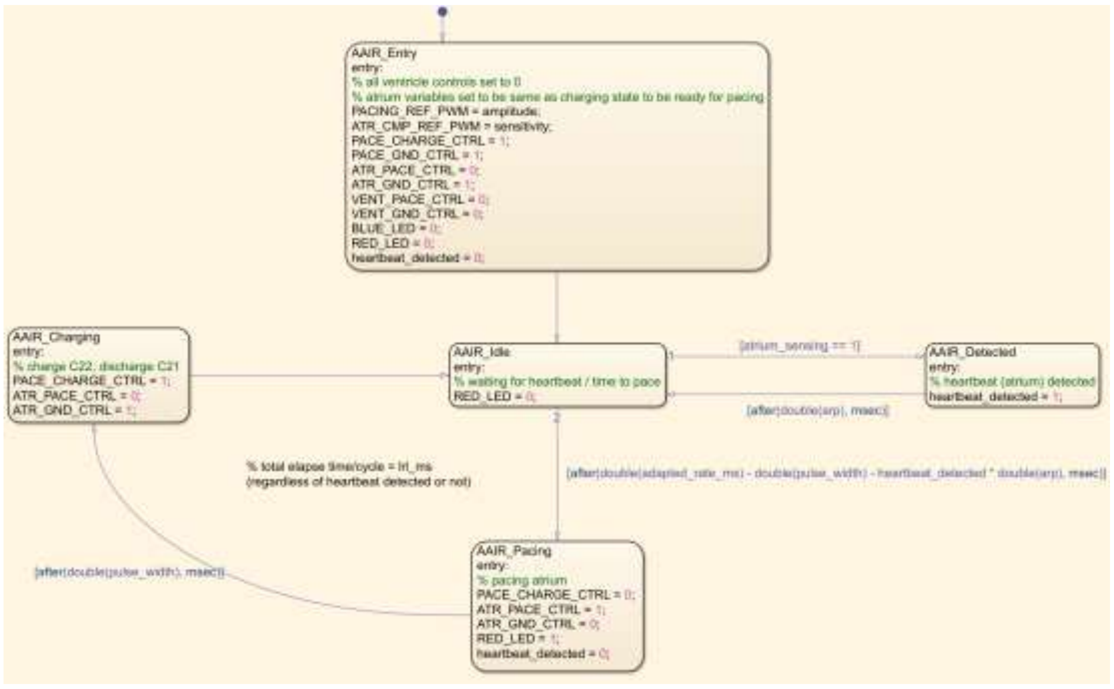
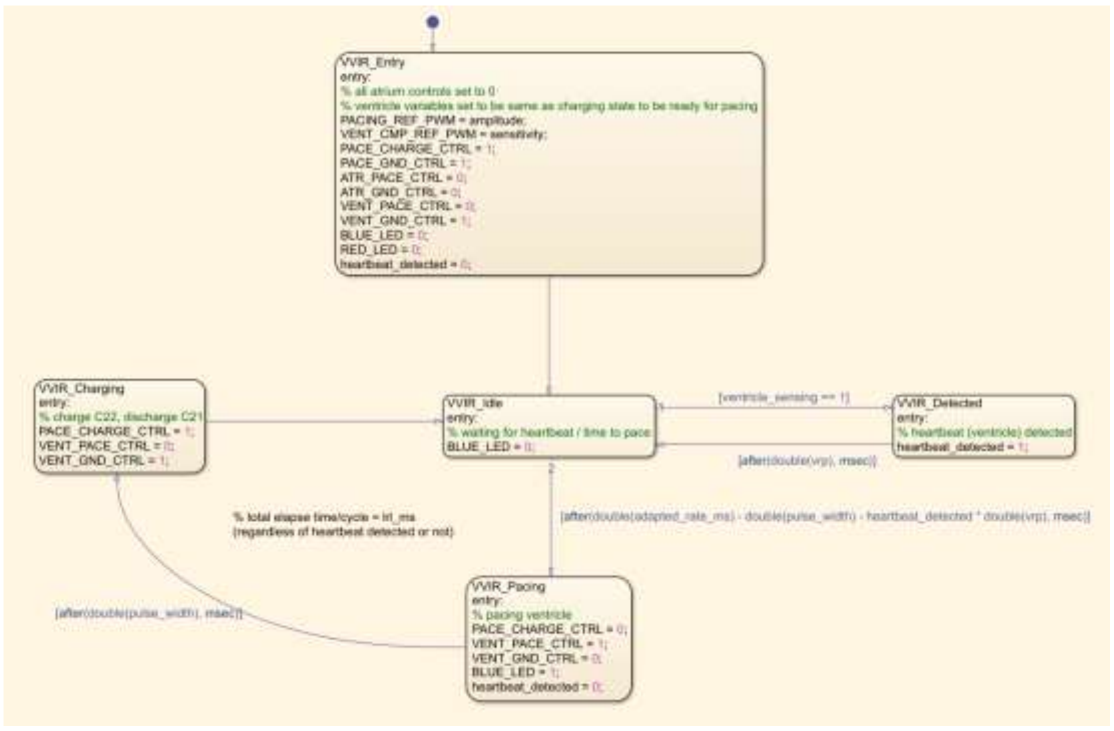

AAIR_Entry
entry:
% all ventricle controls set to 0
% atrium variables set to be same as charging state to be ready for pacing
PACING_REF_PWM = amplitude;
ATR_CMP_REF_PWM = sensitivity;
PACE_CHARGE_CTRL = 1;
PACE_GND_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 1;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 0;
BLUE_LED = 0;
RED_LED = 0;
heartbeat_detected = 0;
AAIR_Charging
entry:
% charge C22, discharge C21
PACE_CHARGE_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 1;
AAIR_Idle
entry:
% waiting for heartbeat / time to pace
RED_LED = 0;
1
[atrium_sensing == 1]
AAIR_Detected
entry:
% heartbeat (atrium) detected
heartbeat_detected = 1;
[after(double(arp), msec)]
2
% total elapse time/cycle = lrl_ms
(regardless of heartbeat detected or not)
[after(double(adapted_rate_ms) - double(pulse_width) - heartbeat_detected * double(arp), msec)]
[after(double(pulse_width), msec)]
AAIR_Pacing
entry:
% pacing atrium
PACE_CHARGE_CTRL = 0;
ATR_PACE_CTRL = 1;
ATR_GND_CTRL = 0;
RED_LED = 1;
heartbeat_detected = 0;

Figure 10. AAIR Submodule in Main



```
VVIR_Entry
entry:
% all atrium controls set to 0
% ventricle variables set to be same as charging state to be ready for pacing
PACING_REF_PWM = amplitude;
VENT_CMP_REF_PWM = sensitivity;
PACE_CHARGE_CTRL = 1;
PACE_GND_CTRL = 1;
ATR_PACE_CTRL = 0;
ATR_GND_CTRL = 0;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 1;
BLUE_LED = 0;
RED_LED = 0;
heartbeat_detected = 0;
```

```
VVIR_Charging
entry:
% charge C22, discharge C21
PACE_CHARGE_CTRL = 1;
VENT_PACE_CTRL = 0;
VENT_GND_CTRL = 1;
```

```
VVIR_Idle
entry:
% waiting for heartbeat / time to pace
BLUE_LED = 0;
```

```
VVIR_Detected
entry:
% heartbeat (ventricle) detected
heartbeat_detected = 1;
```

```
VVIR_Pacing
entry:
% pacing ventricle
PACE_CHARGE_CTRL = 0;
VENT_PACE_CTRL = 1;
VENT_GND_CTRL = 0;
BLUE_LED = 1;
heartbeat_detected = 0;
```

```
[ventricle_sensing == 1]
[after(double(vrp), msec)]
[after(double(adapted_rate_ms) - double(pulse_width) - heartbeat_detected * double(vrp), msec)]
[after(double(pulse_width), msec)]
% total elapse time/cycle = lrl_ms
(regardless of heartbeat detected or not)
```

Figure 11. VVIR Submodule in Main

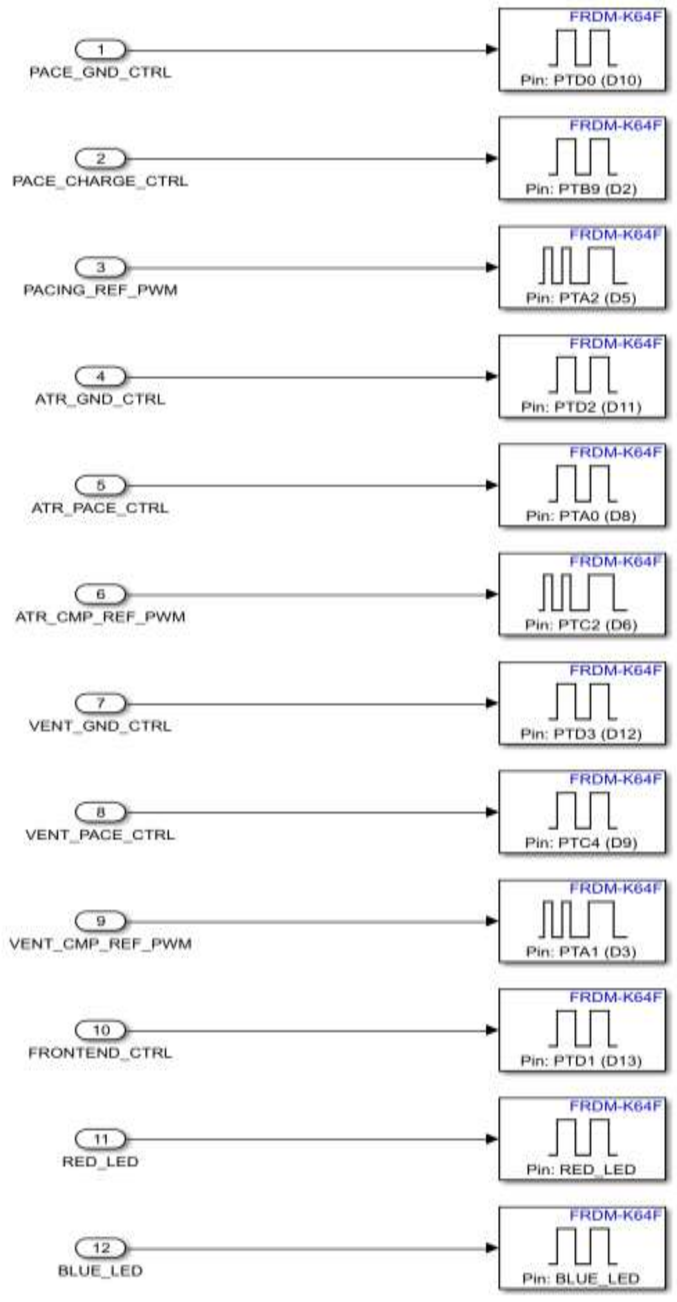


Figure 12. Output Hardware Hiding Module

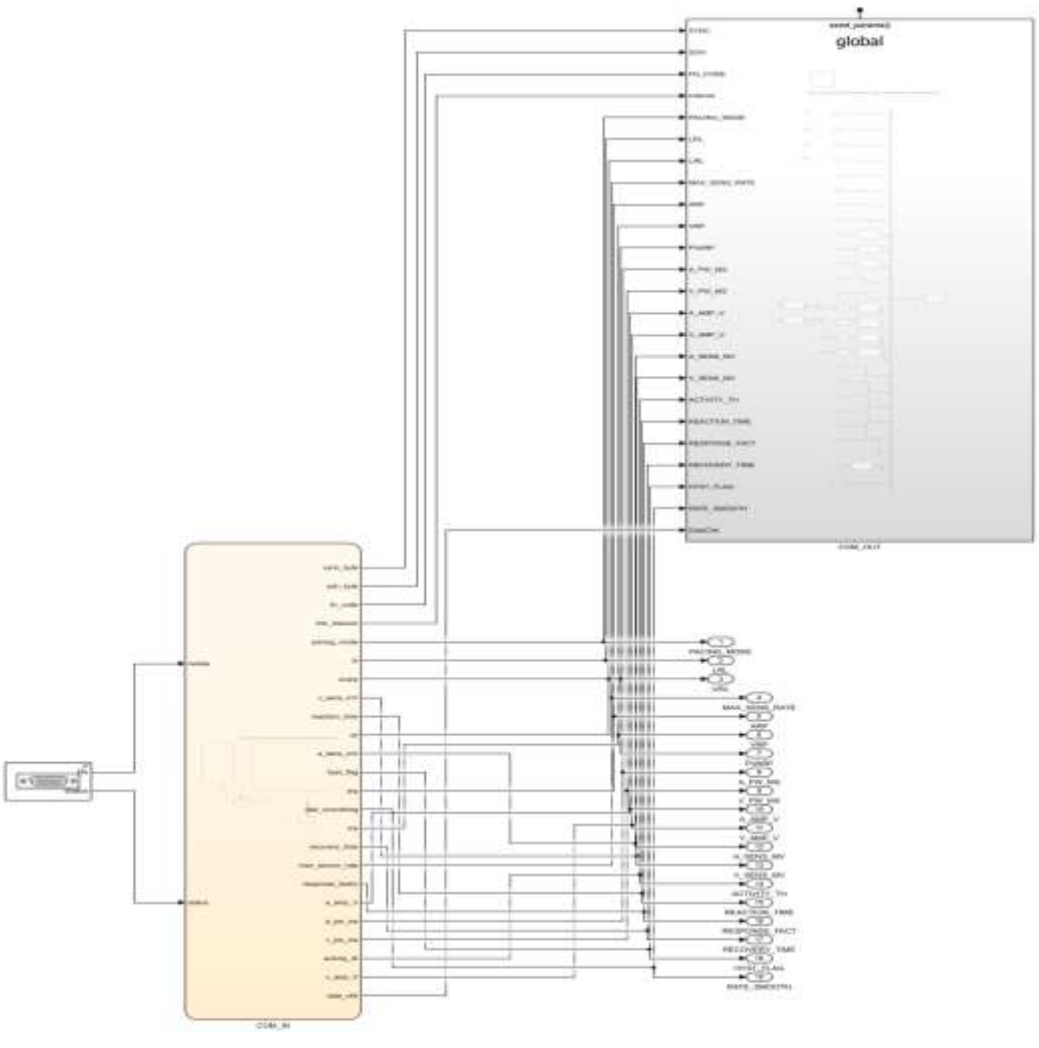
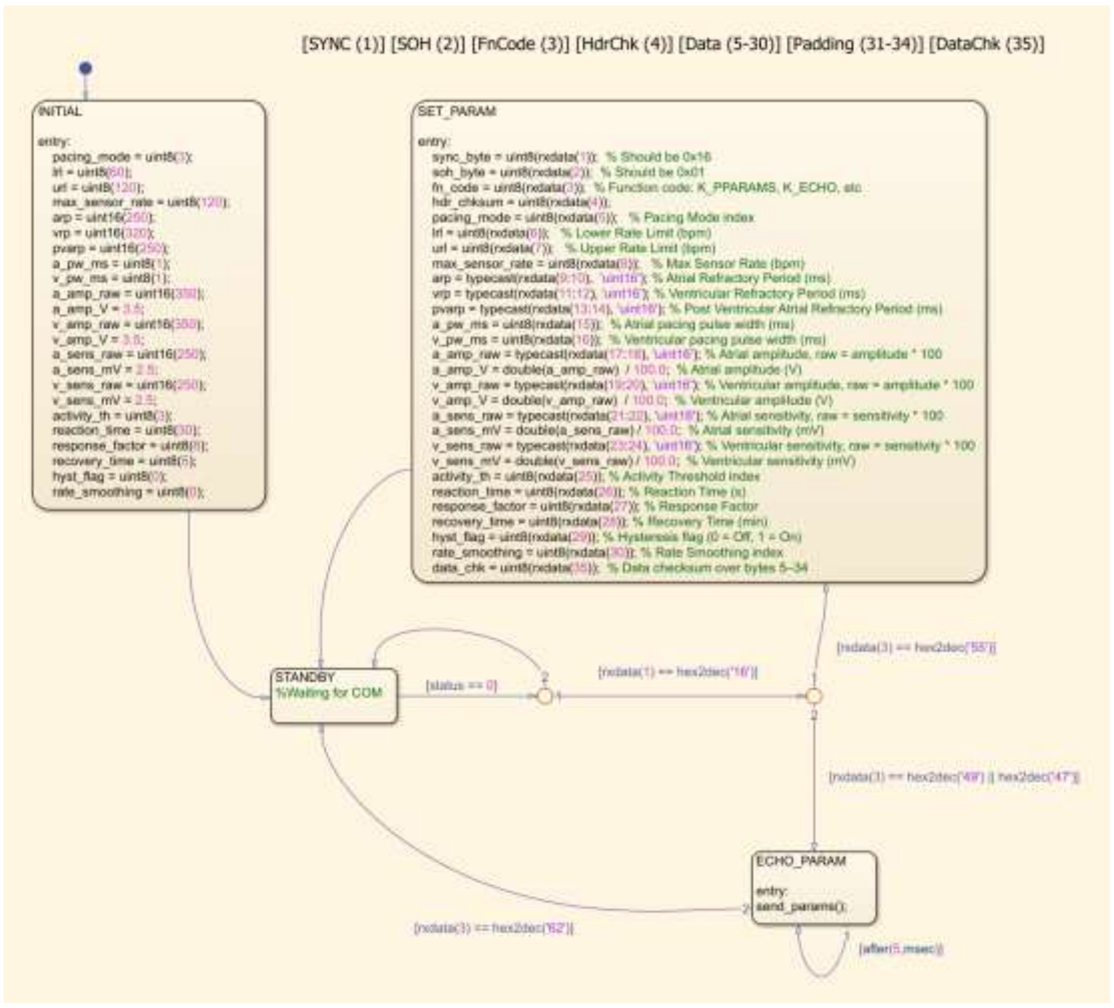


Figure 13. COM Module



[SYNC (1)] [SOH (2)] [FnCode (3)] [HdrChk (4)] [Data (5-30)] [Padding (31-34)] [DataChk (35)]

```
INITIAL

entry:
   pacing_mode = uint8(3);
   lrl = uint8(60);
   url = uint8(120);
   max_sensor_rate = uint8(120);
   arp = uint16(250);
   vrp = uint16(320);
   pvarp = uint16(250);
   a_pw_ms = uint8(1);
   v_pw_ms = uint8(1);
   a_amp_raw = uint16(350);
   a_amp_V = 3.5;
   v_amp_raw = uint16(350);
   v_amp_V = 3.5;
   a_sens_raw = uint16(250);
   a_sens_mV = 2.5;
   v_sens_raw = uint16(250);
   v_sens_mV = 2.5;
   activity_th = uint8(3);
   reaction_time = uint8(30);
   response_factor = uint8(8);
   recovery_time = uint8(5);
   hyst_flag = uint8(0);
   rate_smoothing = uint8(0);
```

```
SET_PARAM

entry:
   sync_byte = uint8(rxdata(1));  % Should be 0x16
   soh_byte = uint8(rxdata(2));  % Should be 0x01
   fn_code = uint8(rxdata(3));  % Function code: K_PPARAMS, K_ECHO, etc
   hdr_chksum = uint8(rxdata(4));
   pacing_mode = uint8(rxdata(5));   % Pacing Mode index
   lrl = uint8(rxdata(6));   % Lower Rate Limit (bpm)
   url = uint8(rxdata(7));   % Upper Rate Limit (bpm)
   max_sensor_rate = uint8(rxdata(8));   % Max Sensor Rate (bpm)
   arp = typecast(rxdata(9:10), 'uint16'); % Atrial Refractory Period (ms)
   vrp = typecast(rxdata(11:12), 'uint16'); % Ventricular Refractory Period (ms)
   pvarp = typecast(rxdata(13:14), 'uint16'); % Post Ventricular Atrial Refractory Period (ms)
   a_pw_ms = uint8(rxdata(15));  % Atrial pacing pulse width (ms)
   v_pw_ms = uint8(rxdata(16));  % Ventricular pacing pulse width (ms)
   a_amp_raw = typecast(rxdata(17:18), 'uint16'); % Atrial amplitude, raw = amplitude * 100
   a_amp_V = double(a_amp_raw)  / 100.0;  % Atrial amplitude (V)
   v_amp_raw = typecast(rxdata(19:20), 'uint16'); % Ventricular amplitude, raw = amplitude * 100
   v_amp_V = double(v_amp_raw)  / 100.0;  % Ventricular amplitude (V)
   a_sens_raw = typecast(rxdata(21:22), 'uint16'); % Atrial sensitivity, raw = sensitivity * 100
   a_sens_mV = double(a_sens_raw) / 100.0;  % Atrial sensitivity (mV)
   v_sens_raw = typecast(rxdata(23:24), 'uint16'); % Ventricular sensitivity, raw = sensitivity * 100
   v_sens_mV = double(v_sens_raw) / 100.0;  % Ventricular sensitivity (mV)
   activity_th = uint8(rxdata(25)); % Activity Threshold index
   reaction_time = uint8(rxdata(26)); % Reaction Time (s)
   response_factor = uint8(rxdata(27)); % Response Factor
   recovery_time = uint8(rxdata(28)); % Recovery Time (min)
   hyst_flag = uint8(rxdata(29)); % Hysteresis flag (0 = Off, 1 = On)
   rate_smoothing = uint8(rxdata(30)); % Rate Smoothing index
   data_chk = uint8(rxdata(35));  % Data checksum over bytes 5-34
```

```
STANDBY
%Waiting for COM
```

```
ECHO_PARAM

entry:
send_params();
```

[status == 0]
[rxdata(1) == hex2dec('16')]
[rxdata(3) == hex2dec('55')]
[rxdata(3) == hex2dec('49') || hex2dec('47')]
[rxdata(3) == hex2dec('62')]
[after(5,msec)]
1
2
1
2
2
1

Figure 114. COM_IN Submodule in COM

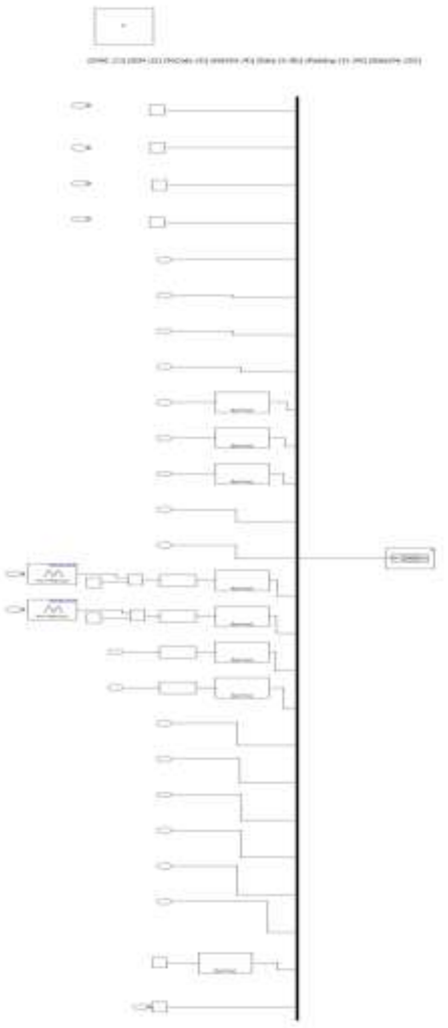


Figure 15. COM_OUT Submodule in COM

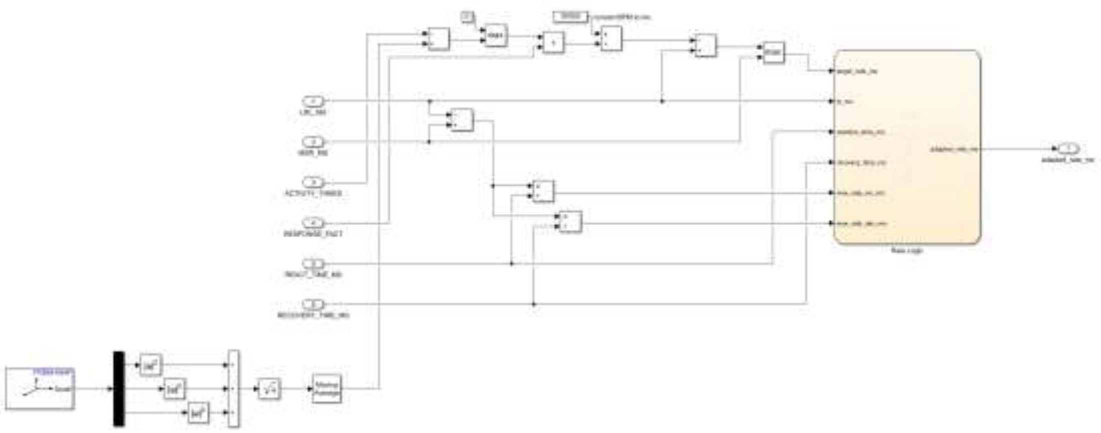


Figure 16. Rate Adaption Submodule in Input Hardware Hiding

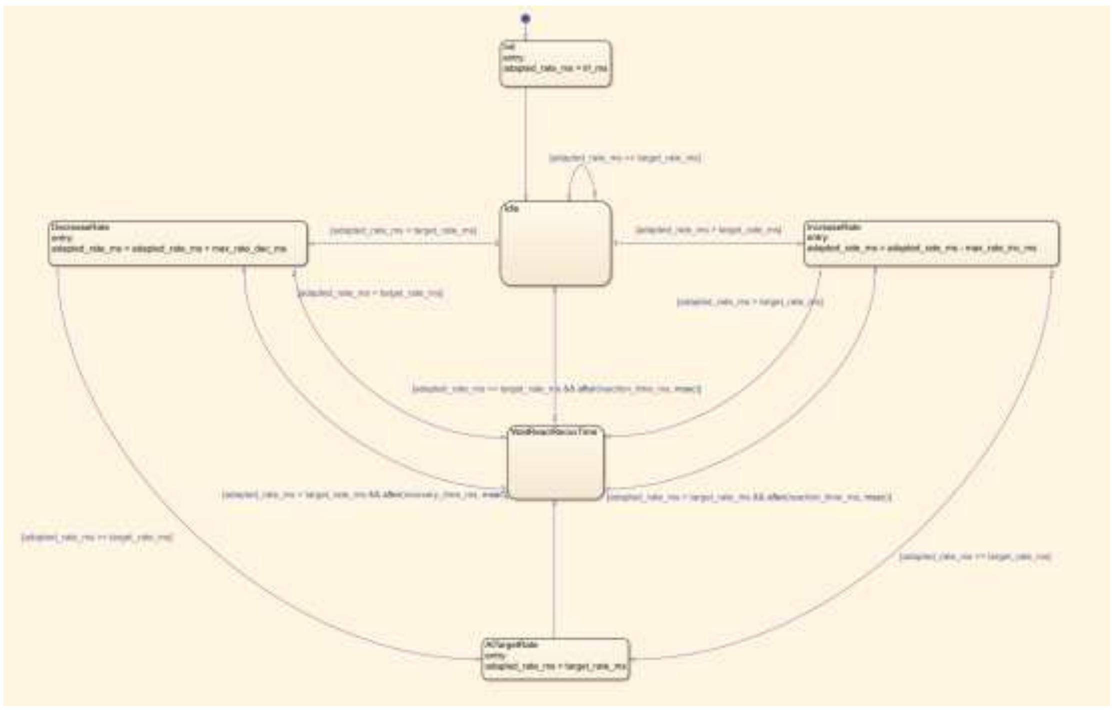

Init
entry:
adapted_rate_ms = lrl_ms
[adapted_rate_ms == target_rate_ms]
Idle
DecreaseRate
entry:
adapted_rate_ms = adapted_rate_ms + max_rate_dec_ms
[adapted_rate_ms < target_rate_ms]
[adapted_rate_ms > target_rate_ms]
IncreaseRate
entry:
adapted_rate_ms = adapted_rate_ms - max_rate_inc_ms
[adapted_rate_ms < target_rate_ms]
[adapted_rate_ms > target_rate_ms]
[adapted_rate_ms == target_rate_ms && after(reaction_time_ms, msec)]
WaitReactRecovTime
[adapted_rate_ms < target_rate_ms && after(recovery_time_ms, msec)]
[adapted_rate_ms > target_rate_ms && after(reaction_time_ms, msec)]
[adapted_rate_ms >= target_rate_ms]
[adapted_rate_ms <= target_rate_ms]
AtTargetRate
entry:
adapted_rate_ms = target_rate_ms

Figure 17. Rate logic Submodule in Rate Adaption Submodule

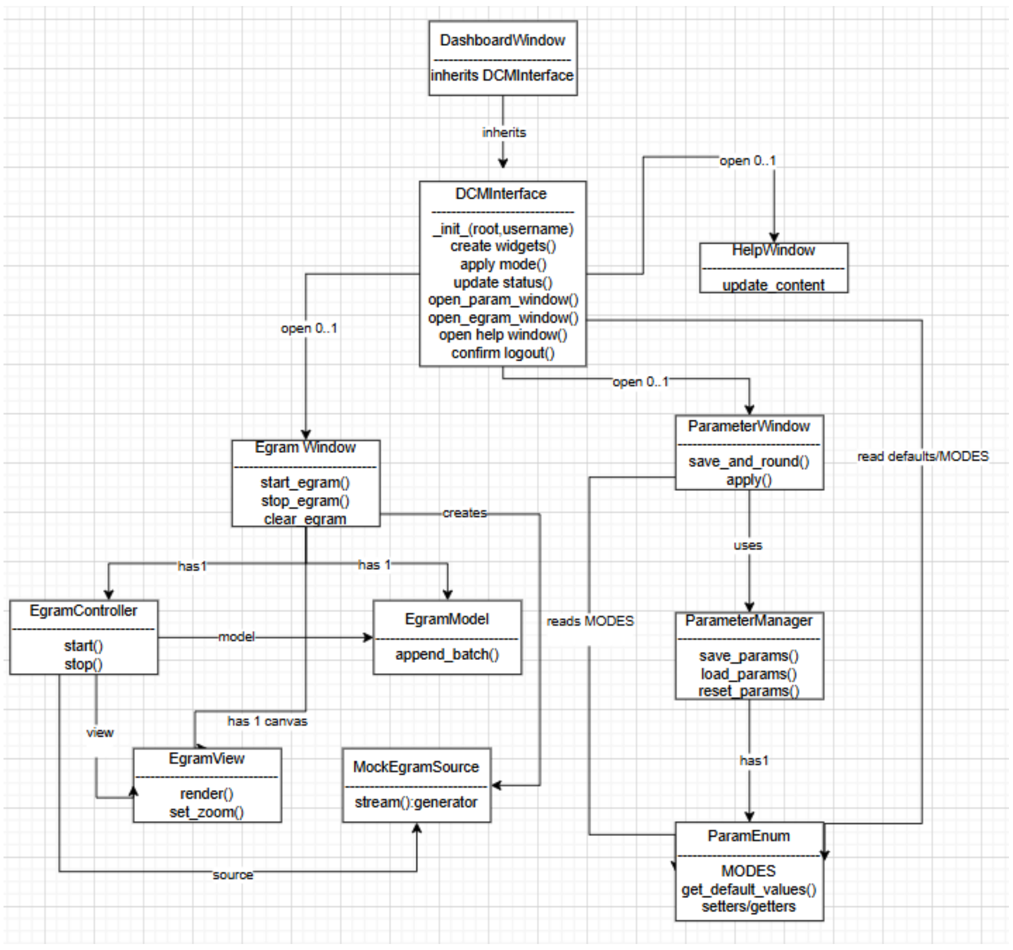
2.3.6 DCM Overview & System Architecture

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.

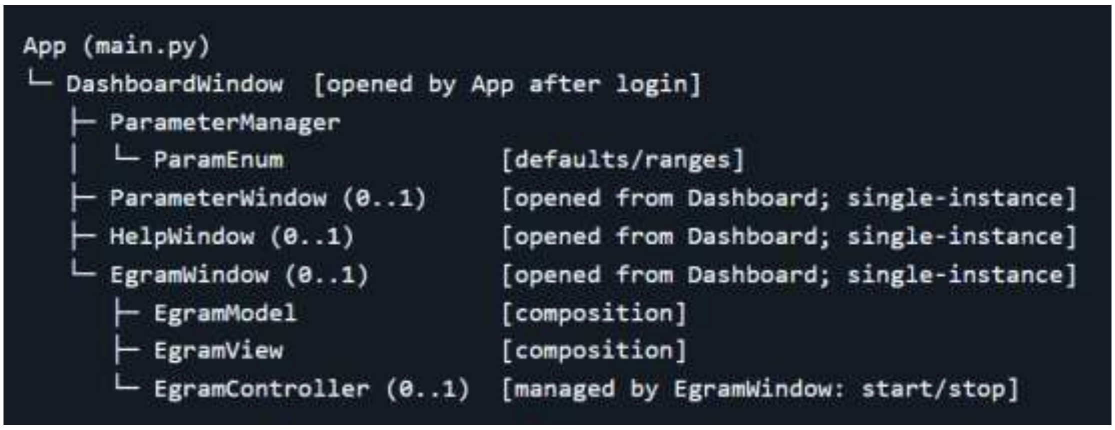
Module	Purpose / Responsibility
auth.py	Manages user registration, login, and logout
dashboard.py	Main post-login hub. Provides access to mode selection, parameter editing, EGram viewing, and logout.
mode_config.py	Defines pacing modes and programmable parameters through the ParamEnum class.
ParamOps.py	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



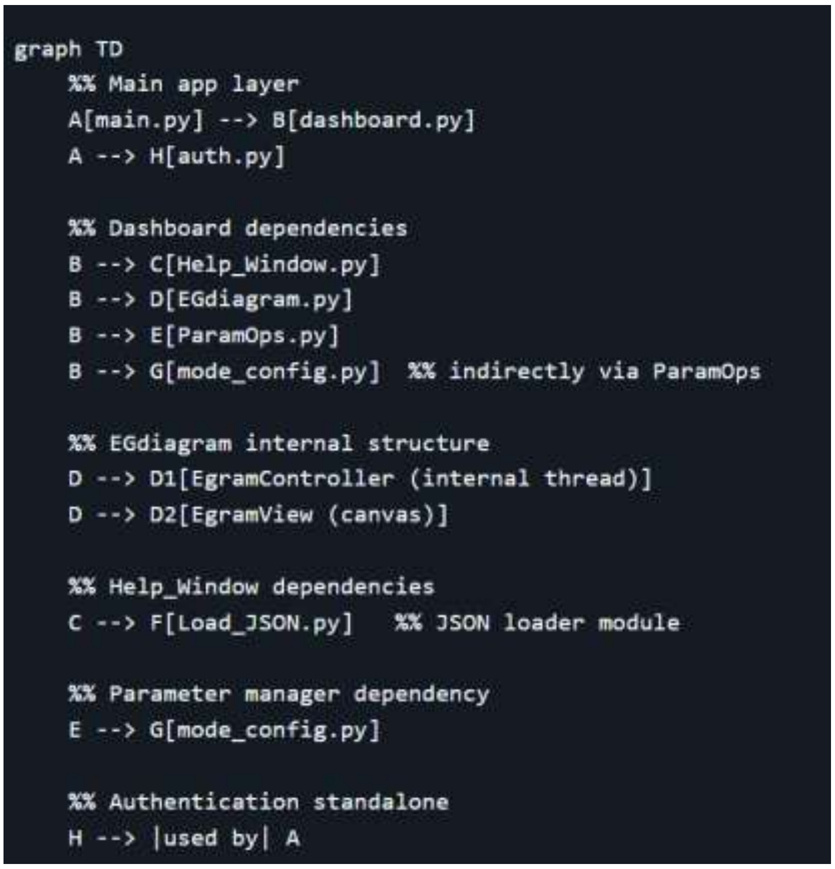
Has 1 marks composition, the parent makes and strictly owns that part, like DCM interface to ParameterManager, EgramWindow to Model, Controller/View. But these are all only one thing, and it is tied to the parent. When the parent is destroyed, so are the other parts. By contrast, open 0..1 marks aggregation 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
```

2.3.6.1 Deliverable 2

Procedure in Deliverable 2 expands on the existing DCM and provides full serial communication for the real pacemaker device. In the communication subsystem two modules are built: Serial_Manager.py which allows low level byte-based serial I/O and Communication.py which provides a high-level protocol defined for that project.

Overall Responsibilities

• Find Serial Ports and choose the device.

• Setup and terminate a stable serial connection

• Package all programmable parameters in the 29-byte frame format requested.

• Submit the frame to pacemaker and wait for an acknowledgment.

• Ask for device-stored parameters and ensure that they correspond to the doctor's input.

• For on-line visualization, stream atrial/ventricular Egram samples from the pacemaker.

• Update UI with connection status, noise, out-of-range conditions, or new-device conditions.

Communication Architecture (Low-Level Adapter)

A SerialManager Module

SerialManager includes all direct communication to the operating system's serial interface. This module will always keep simple to respect the rule of information hiding, without any higher-level logic, which interacts with the OS serial API.

This provides:

- open_port(port_name)
- close_port ()
- write_bytes (data: bytes)
- read_bytes (n: int)

PacemakerCommunication (High-Level Protocol Layer)

This specification characterizes the communication protocol between the DCM and the pacemaker. It includes logic for:

- Enumerate ports (list_ports ())
- Connecting/disconnecting
- Programmable parameters are encoded into the agreed 29-byte message frame
- Receiving acknowledgments by sending parameter frames
- Reading the device ID
- Stored parameters need to be verified
- Egram window for streaming Egram samples

Parameter transmission workflow (For four steps)

This workflow makes sure that the Pacemaker can always use the exact configuration provided by doctors.

1. Parameter Normalization (UI Layer)

Using ParamEnum, all parameter values are collected in the ParameterWindow, and it validates them (range checking, step rounding, cross-constraints).

2. Encoding (Communication Layer)

The string format is <BBBBHHHBBHHHHBBBBBB4x>.

The total length of the code string is 4+30+1=35 bytes of length. Thefirst four are headers, which are SYNC, SOH, FnCode and HdrChk. The SYNC and SOH are fixed at 0x16 and 0x01, the FnCode is used to execute the corresponding functions, 0x49 is request parameters, 0x55 is sending params, 0x47 is starting Egram, 0x62 is stop Egram. The HdrChk is not implement at there since not enough time, but for security assurance, we will implement if have enough time to do the XOR operation with the first 3 headers to make sure the communication is legal and complete.

The last byte is Data check byte, which is not implement in D2 since do not have enough time. The conceptual function of it is to check the data length and the correctness (i.e. whether is legal and fulfill the range and stepping rules).

Following the reference to D2 above, the validated parameters are encoded in a fixed

29-byte frame as provided in the specification of D2:

- Byte 0: Mode
- Bytes 1–3: LRL, URL, MSR. (B means 1 byte)
- Bytes 4–9: Timing parameters (ARP, VRP, PVARP)
- Bytes 10–11: Pulse widths. (H means 2 bytes)
- Bytes 12–19: Amplitudes / Sensitivity (scaled integer formats). (H means 2 bytes)
- Bytes 20–25: Rate-adaptive parameters
- Remaining bytes reserved / padding

3. Transmission

A frame is sent via SerialManager.write_bytes (). The device responds with an acknowledgment frame with its internal stored values.

4. Verification

The DCM compares device-stored parameters with the doctor entered values. The UI can prevent “Apply” if the match is misaligned and it then instructs the user to upload the parameters once more.

Egram Data Streaming.

As DCM plots the Egram diagram, the corresponding binary frames have been continuously received which contain atrial and ventricular analog samples. The EgramController executes a timed loop.

- Reads raw samples from PacemakerCommunication.get_egram_sample ().
- Buffers and forwards them to the EgramView.
- Detects disconnect, noise, or unstable communication in real time and stops the plotting safely.

Safety Considerations

The communication layer contributes to several safety requirements.

• Prevents applying incorrect parameters through strict verification of device-stored values.

• Detects new devices by comparing the stored last_connected_device to the device ID obtained from the pacemaker.

• Prevents unsafe UI actions (e.g., blocking “Load to Pacemaker” when disconnected).

• Handles noisy or unstable connections by displaying warnings and disabling Egram updates.

Thus, these design choices help ensure the DCM never silently loads incorrect or unsafe pacing configurations.

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. New for D2: There is a new button called “Load to Pacemaker” to send parameters to Pacemaker, and block upload when disconnected.	Add presets/bulk import–export; inline validation tips; support future modes/extra fields without UI rewrites. New for D2: This Parameter window can add presets and export functions, also can add device-side logs.
Egram	Start/Stop/Clear; live plotting	Swap to serial adapter; add

	with mock source (D1); guard double-Start; disable Clear while running. New for D2: New version displays real-time Egram from serial data and stops plotting on disconnect and noise.	channel(D1) filters/zoom/export (PNG/CSV); performance throttling and annotations. New for D2: New version adds zoom, filter, and export options.
DashboardWindow	Navigation hub; open/lift single instances of Help or Egram or Parameter windows; show session/status. New for D2: New design shows available COM ports, shows connecting or disconnecting to Pacemaker, and shows device ID after connects status.	Role-based buttons; quick mode/parameter presets; session banner and status toasts. New for D2: The new part adds advanced status indicators and supports several remembered devices.

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.
Communication.py (D2)	This provides high level serial communication, which supports encode and transmit parameter frame. It can also read device ID and stored parameters, also with streaming Egram data	Add checksum and error recovery, with async and queued communication.
Serial_Manager.py (D2)	This code provides low level serial port control, for open, close, read and write. It also handles timeouts and device errors.	It can automatically connect with board channel and read buffers.

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. Processes programmable parameters through use of saturation functions and unit conversions. **New for d2:** Integrate accelerometer signal processing and include rate adaptation to compute adapted rate based on activity levels.	Update for new pin mappings, hardware upgrades, or added input devices. **New for d2:** add diagnostic flags to detect sensor faults or signal integrity issues, implement more advanced filtering algorithms to further reduce noise and unwanted motions in accelerometer signals
Main module	Controls selection and logic for all pacing modes. Coordinates programmable parameters and signal flow. **New for d2:** Add the four new rate-adaptive modes; integrate rate adaptation logic with reaction time and recovery time smoothing; implement dynamic LRL adjustment based on MSR and activity levels.	Add new pacing modes, expand parameter sets, or refactor for modularity. **New for d2:** add fallback mode logic in case rate adaption fails, implement mode-switching safety checks
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. **New for d2:** add diagnostic to detect pacing capture

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. **New for d2:** add ventricular safety pacing windows to prevent pacing into refractory periods
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. **New for d2:** Implement auto-threshold detection to dynamically adjust sensing threshold based on signal amplitude.
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. **New for d2:** Add safety margins for sensing thresholds to prevent under sensing of weak signals.
Output Hardware Hiding module	Maps logic outputs to physical pins for actuation. Isolates main logic from hardware dependencies. **New for d2:** Route atrial and ventricular egram signals to DCM; ensure pacing event markers are in sync with egram data transmission.	Update routing for new hardware, integrate confirmation feedback, or add outputs. **New for d2:** Add output signal integrity checking and fault detection, add programmable output delay or jitter to support specific hardware timing requirements.

AOOR Submodule in main	Delivers atrial pacing pulses at a rate that adapts to activity. Paces at adapted rate computed from accelerometer data, bounded between LRL and MSR.	Verify rate response curve accuracy, validate reaction/recovery time implementation, test transitions between fixed and adaptive rates.
VOOR submodule in main	Delivers ventricular pacing pulses at a rate that adapts to patient activity. Paces at adapted rate computed from accelerometer data, bounded between LRL and MSR.	Verify rate response curve accuracy; validate reaction/recovery time implementation; test transitions between fixed and adaptive rates.
AAIR submodule in main	Senses intrinsic atrial activity and inhibits pacing if detected within adapted pacing interval. Adapts pacing rate to activity while maintaining sensing and inhibition logic.	Adjust sensing thresholds with rate adaptation; validate ARP refractory timing under adaptive conditions; test mode transitions.
VVIR submodule in main	Senses intrinsic ventricular activity and inhibits pacing if detected within adapted pacing interval. Adapts pacing rate to patient activity while maintaining sensing and inhibition logic.	Adjust sensing thresholds with rate adaptation; validate VRP refractory timing under adaptive conditions; test mode transitions; enhance error handling for dynamic rate scenarios.
COM module	Manages bidirectional serial communication between pacemaker and DCM. Receives programmable parameters and commands from DCM (COM_IN) Transmits device status, egram data, and pacing events to DCM (COM_OUT).	Handle packet buffering and multiplexing; ensure real-time egram streaming without blocking critical pacing logic; add support for protocol version updates or expanded basic data.

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.

Design Decision for D2:

Serial Communication Architecture

In our Deliverable 2, we implemented a two-layer communication architecture that included SerialManager (low-level) and PacemakerCommunication (high-level). That design applies information-hiding concepts: the low-level module separates OS-specific serial behavior, and the high-level module provides clean APIs for connection, parameter transmission, device verification, and Egram streaming. It makes it much easier to maintain while you have the potential to easily implement changes to protocol later in the future without touching the UI layer.

Structured Parameter Frame

For Parameter Structure. All programmable parameters were encoded in a fixed-size 29-byte frame. This decision allows deterministic parsing on both ends while mitigating transmission errors and enables rapid verification through comparison of device-stored values with DCM values. A scaled integers approach ensures precision and compactness for amplitudes and sensitivities

Logic of Uploading ParameterWindow

The ParameterWindow was then extended with the "Load to Pacemaker" operation. This ensures that the parameters do go through UI validation, are passed into the frame, and will be only enforced if there is a stable serial connection. When mismatches arise, blocking Apply ensures that unsafe pacing settings are avoided and the end-to-end correctness of parameters is enforced.

Dashboard Connections & Device Identity and Device ID

In this regard, Dashboard provides the central place for discovering, connecting, and checking the identity of the device. It makes all traces easier to see and the device ID can be stored as well as reminds the user when a new device is detected, so that you don't have silent misconfiguration. Displaying connection quality like disconnected, noise, out-of-range supports more secure real-time monitoring.

Real-Time Egram Streaming

EgramWindow was redesigned for real-time data as it was streamed from PacemakerCommunication instead of dummy data like D1. This decision enables clinicians to see the behavior of real pacing and the quality of signals. Egram diagram can automatically stop if there is misleading visualization caused by disconnection and noise. As a result, safety can be kept.

UI-State Synchronization

Throughout the DCM, state flags such as saved, applied, device-synced were added to prevent invalid actions.

This reduces ambiguous of UI, which can improve robustness and make sure that doctors know the device shows configuration we hope.

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. Process all inputs	Adapt for new boards, sensors, or pin mappings. Add preprocessing to for example like noise filtering. **New for d2:** Implement multi-rate filtering for accelerometer data, add

	(parameters, heart signals, accelerometer) through unit conversion and saturation functions before passing to Main module.	adaptive filter coefficients based on patient activity pattern
Main module	Centralize control/selection of pacing mode. Parameterize for easy updates. Separate each pacing mode into dedicated submodules to simplify debugging and testing.	Expand with new pacing modes. Integrate more complex parameter sets. **New for d2:** Implement mode transition state machine to manage smooth transitions between fixed-rate and rate-adaptive modes, add redundancy checks to ensure only one mode is active at a time
AOO Submodule in main	Deliver fixed-rate atrial pulses, independent of sensing. Use dedicated charging and pacing states for cycling at the programming interval.	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. Isolate main control logic from hardware dependencies using pin-to-signal mapping.	Accommodate new output hardware. Add feedback/confirmation or expandable outputs.
AOOR Submodule in main	Implement rate-adaptive atrial pacing using adapted_rate_ms from Input Hardware Hiding. Apply reaction time and recovery time smoothing to avoid abrupt rate changes.	Test edge cases such as rapid activity transitions and accelerometer noise. Implement fallback to AOO if rate adaptation signals become invalid. Add diagnostics to report current adapted rate and activity level.
VOOR Submodule in main	Implement rate-adaptive ventricular pacing using adapted_rate_ms from Input Hardware Hiding. Apply reaction time and recovery time smoothing to avoid abrupt rate changes.	Test edge cases such as rapid activity transitions and accelerometer noise. Implement fallback to AOO if rate adaptation signals become invalid. Add diagnostics to report current adapted rate and activity level.
AAIR Submodule in main	Implement rate-adaptive atrial demand pacing. Combine AAI sensing logic with adapted_rate_ms-based timing. Pace only if no intrinsic event sensed within adapted interval. Use separate sensing, pacing, charging, and idle states.	Implement safety checks for mode switching with active rate changes. Test transitions between activity levels while maintaining sensing integrity. Implement mode-specific fallback.
VVIR Submodule in main	Implement rate-adaptive ventricular demand pacing. Combine VVI sensing logic with adapted_rate_ms-based timing.	Implement mode specific fallback Implement safety checks for mode switching with active rate changes.

	Pace only if no intrinsic event sensed within adapted interval. Use separate sensing, pacing, charging, and idle states.	Test transitions between activity levels while maintaining sensing integrity.
COM module	Implement bidirectional serial communication handler. Separate COM_IN which receives parameters from DCM and COM_OUT which transmit status and egram to DCM into distinct functional blocks.	Ensure real-time egram streaming without blocking critical pacing logic. Consider changing serial port to wireless communication protocols in the future.

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.

MG for D2:

Communication Layer

A new communication layer is introduced in Deliverable 2, consisting of:

• SerialManager – responsible for low-level serial I/O (open/close/read/write).

• PacemakerCommunication – high-level protocol layer built on top of SerialManager.

Provides port enumeration, connection handling, parameter-frame transmission, device-ID retrieval, and real-time Egram streaming.

This layer is used by:

• DashboardWindow for port listing, connect/disconnect, and device identification.

• ParameterWindow for uploading programmable parameters to the pacemaker and verifying device-stored values.

• EgramWindow for requesting and streaming real-time Egram data from the device.

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.

New for D2:

• refresh_ports().

• connect_device().

• disconnect_device().

• update_connection_status().

Extended Responsibilities: Display port list, connection state, device ID, and "new device detected" warnings

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.
- New for D2:
    - load_to_pacemaker() — Upload parameters to the device.
    - compare_with_device() — Verify stored values before Apply.

Extended Behaviour: Block upload when disconnected; enforce local-vs-device consistency.

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.

New for D2:

• start_egram() — start real-time streaming.

• stop_egram() — stop streaming.

• handle_disconnect() — stop automatically if communication is lost.

Extended Behaviour: Plots real pacemaker samples instead of mock data.

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.

New MIS for D2

PacemakerCommunication — Interface Specification.

Purpose: High-level communication protocol between the DCM and the pacemaker.

Public Methods:

- list_ports() -> list[str].
- connect(port_name: str) -> bool.
- disconnect() -> None.
- is_connected() -> bool.
- get_device_id() -> str.
- upload_parameters(frame: bytes) -> bool.
- read_device_parameters() -> dict.
- get_egram_sample() -> tuple[int, int] (atrial, ventricular)

SerialManager— Interface Specification.

Purpose: Low-level serial I/O abstraction.

Public Methods:

- open_port(port: str) -> bool.
- close_port() -> None.
- write_bytes(data: bytes) -> None.
- read_bytes(n: int) -> bytes.
- is_open() -> bool.

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.
- New for D2:
    - On “Refresh”, calls comm.list_ports() and updates dropdown.
    - On “Connect”, calls comm.connect(), retrieves device ID, updates UI labels.
    - Compares retrieved device ID with the stored one to trigger “New Device Detected”.
    - Continuously updates connection-quality indicators using comm-reported flags.

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.
- New for D2:
  - On "Load to Pacemaker", retrieves validated parameters, calls comm.upload_parameters().
  - After upload, retrieves device-stored parameters and compares with local values.
  - If mismatch is detected, Apply is blocked and a warning dialog is shown.
  - Maintains three states: Local Saved → Device Uploaded → Verified.

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.
- New for D2:
    - Creates a controller loop that periodically calls comm.get_egram_sample().
    - Converts received raw samples into displayable waveform points.
    - Automatically stops plotting when:
        - device disconnects,
        - data is unstable,
        - noise flags are raised.
    - Notifies DashboardWindow of connection issues.

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.

New for D2:

PacemakerCommunication

- Retains a SerialManager instance for all low-level communication.
- Encodes programmable parameters into the 29-byte protocol frame.
- Confirms acknowledgment frames from the pacemaker.
- Stores the last connected device ID to detect device changes.
- Runs a loop to fetch egram samples when requested.
- Manages connection drops by alerting UI modules.

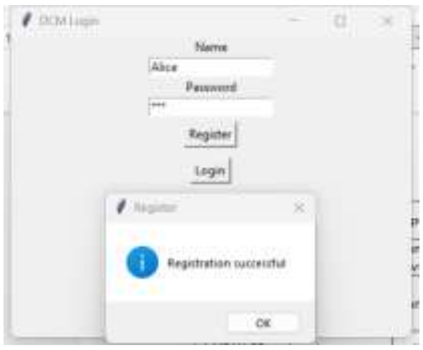
SerialManager

- Wraps pyserial operations.
- Uses blocking reads with timeout to prevent UI freezing.
- Converts raw bytes into Python byte streams.
- Reports port failures or disconnect events to PacemakerCommunication.

3.4 Testing

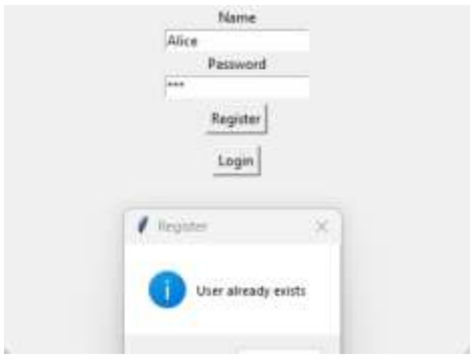
3.4.1 AUTH Testing

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



Registration successful

Figure 18. Trial 1: Register New User



User already exists

Figure 19. Trial 2: Prevent Duplicate Registration

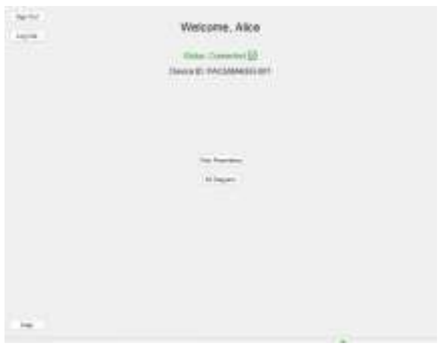
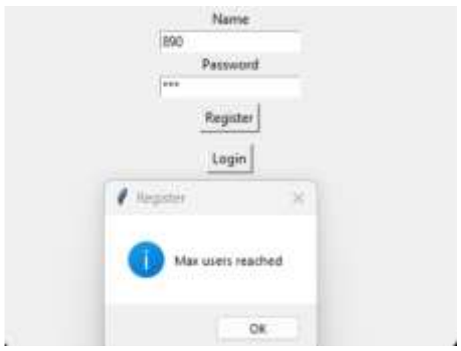

Welcome, Alice

Figure 20. Trial 3: Login Success

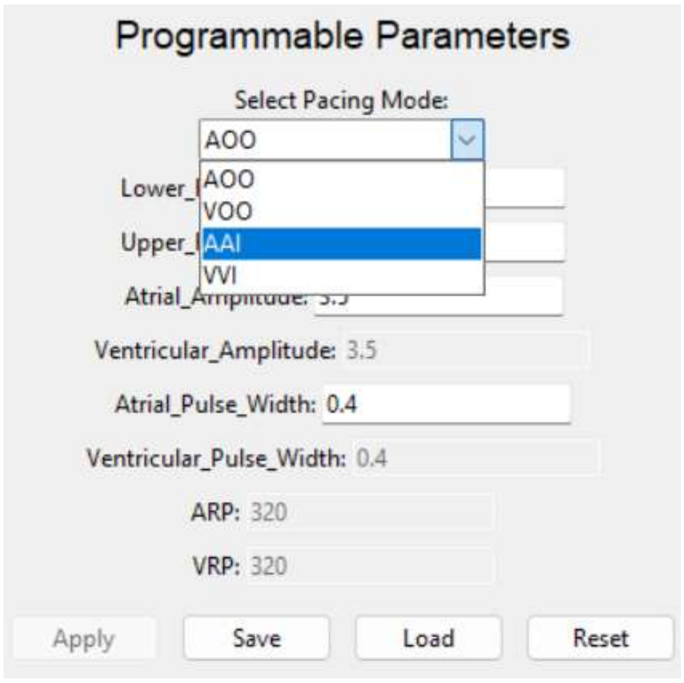


Max users reached

Figure 21. 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



Programmable Parameters

Select Pacing Mode: AOO

Lower_

Upper_

Atrial_Amplitude: 3.5

Ventricular_Amplitude: 3.5

Atrial_Pulse_Width: 0.4

Ventricular_Pulse_Width: 0.4

ARP: 320

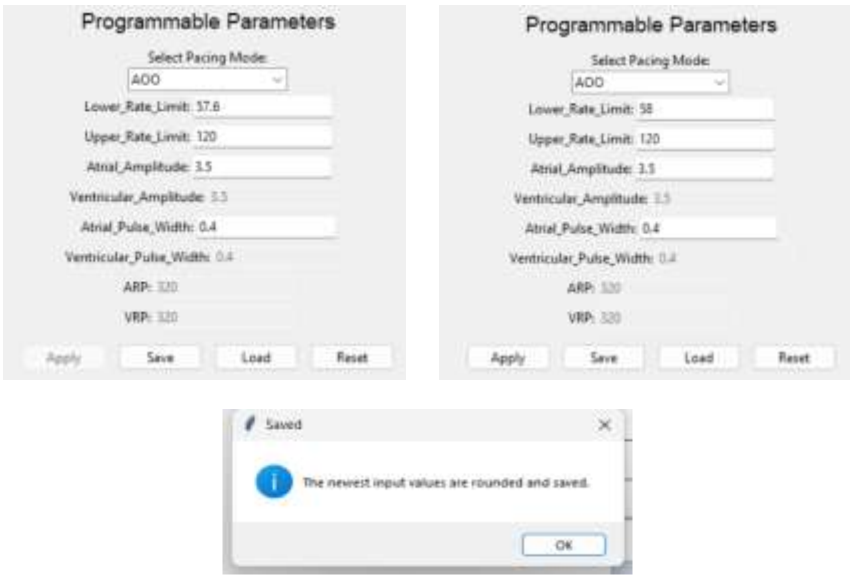
VRP: 320

Figure 22. 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	AA=3.3 V →	Error dialog; not	Fig3	Pass

	amplitude gap	Save	applied		
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


Programmable Parameters

Select Pacing Mode: AOO
Lower_Rate_Limit: 57.6
Upper_Rate_Limit: 120
Atrial_Amplitude: 3.5
Ventricular_Amplitude: 3.5
Atrial_Pulse_Width: 0.4
Ventricular_Pulse_Width: 0.4
ARP: 320
VRP: 320

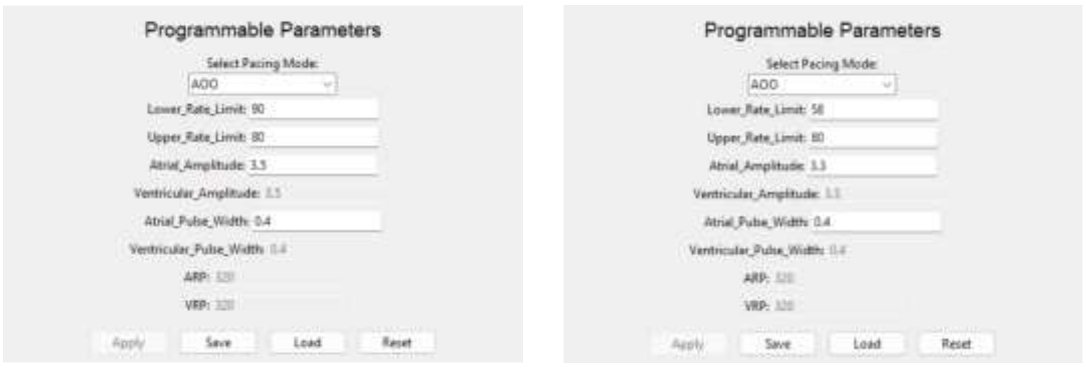
Programmable Parameters

Select Pacing Mode: AOO
Lower_Rate_Limit: 58
Upper_Rate_Limit: 120
Atrial_Amplitude: 3.5
Ventricular_Amplitude: 3.5
Atrial_Pulse_Width: 0.4
Ventricular_Pulse_Width: 0.4
ARP: 320
VRP: 320

Saved

The newest input values are rounded and saved.

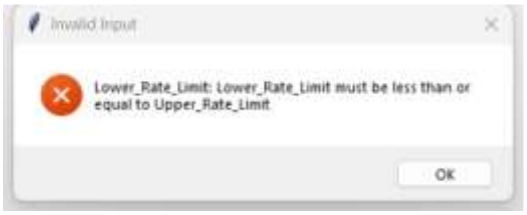
Figure 23. Trial 1: Step Rounding (LRL)


Programmable Parameters

Select Pacing Mode: AOO
Lower_Rate_Limit: 90
Upper_Rate_Limit: 80
Atrial_Amplitude: 3.5
Ventricular_Amplitude: 3.5
Atrial_Pulse_Width: 0.4
Ventricular_Pulse_Width: 0.4
ARP: 320
VRP: 320

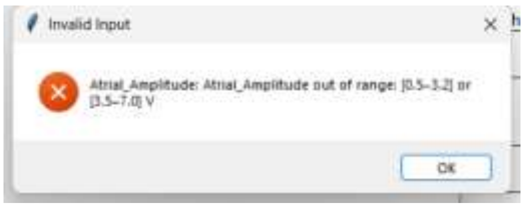
Programmable Parameters

Select Pacing Mode: AOO
Lower_Rate_Limit: 58
Upper_Rate_Limit: 80
Atrial_Amplitude: 3.5
Ventricular_Amplitude: 3.5
Atrial_Pulse_Width: 0.4
Ventricular_Pulse_Width: 0.4
ARP: 320
VRP: 320


Invalid Input

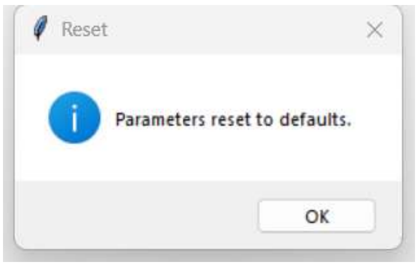
Lower_Rate_Limit: Lower_Rate_Limit must be less than or equal to Upper_Rate_Limit

Figure 24. Trial 2: Cross-constraint


Invalid Input

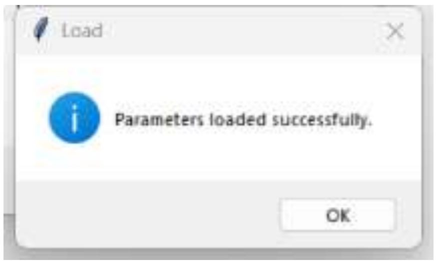
Atrial_Amplitude: Atrial_Amplitude out of range: [0.5–3.2] or [3.5–7.0] V

Figure 25. Trial 3: Invalid Amplitude Gap


Reset

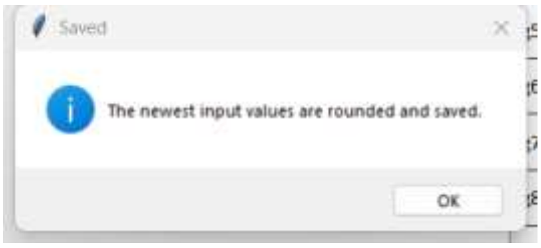
Parameters reset to defaults.

Figure 26. Trial 4: Reset to Defaults


Load

Parameters loaded successfully.

Figure 27. Trial 5: Load Refresh


Saved

The newest input values are rounded and saved.

Figure 28. 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

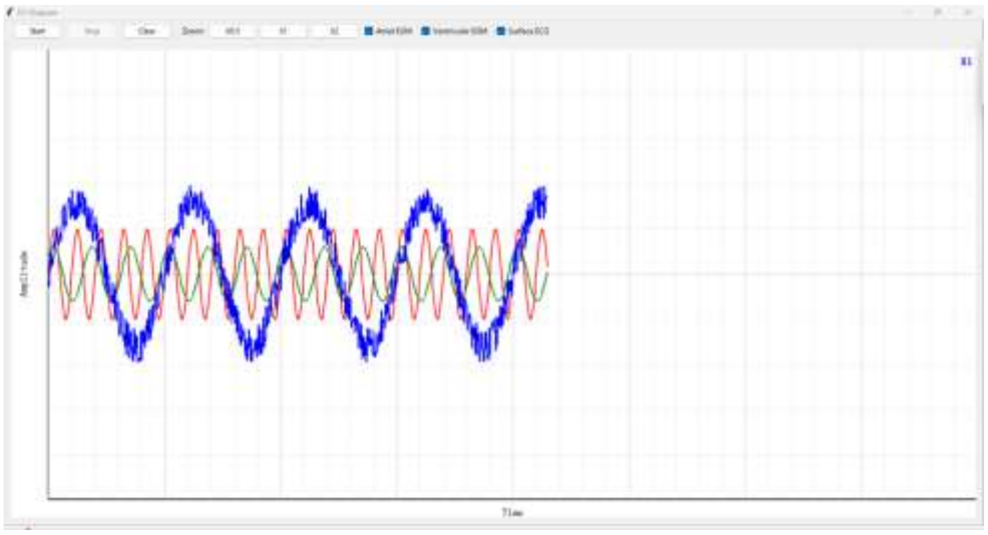
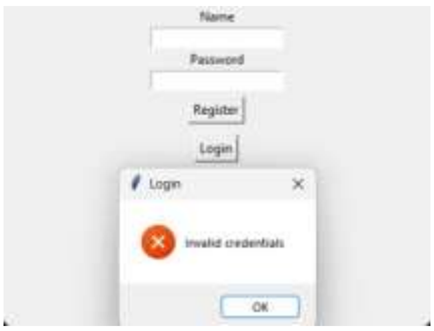


Figure 29. Trial 1: Start Plotting

Figure 30. 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



Invalid credentials

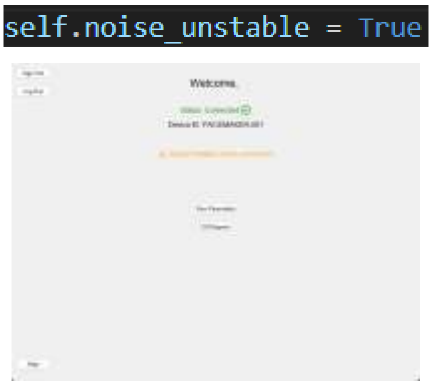
Figure 31. Trial 1: Missing Docs Fallback

```
self.is_connected = False
```

Figure 32. Trial 2: Self_connect is false

```
self.out_of_range = True
```

Figure 33. Trial 3: Self out of range is true



```
self.noise_unstable = True
```

Figure 34. Trial 4: Self_noise unstable is true

Serial Communication Testing (D2)

Trial 1 – Port Refresh

Purpose:

Verify that the DCM correctly detects available serial ports and updates the list after clicking *Refresh*.

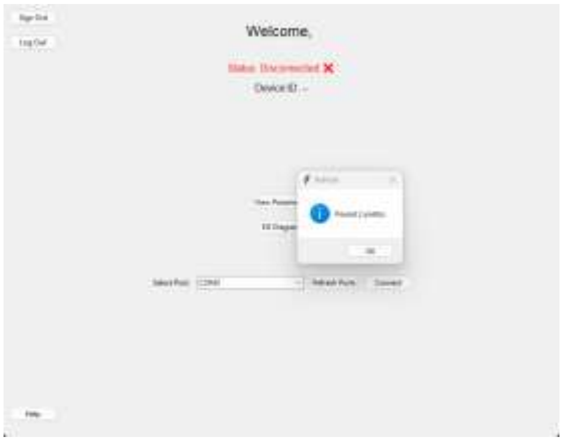
System Input:

Disconnect and reconnect the pacemaker USB device; click Refresh Ports.

Expected Output:

A message dialog appears showing the updated number of detected ports like "Found 1 or 2 port(s), and the port dropdown list refreshes accordingly.

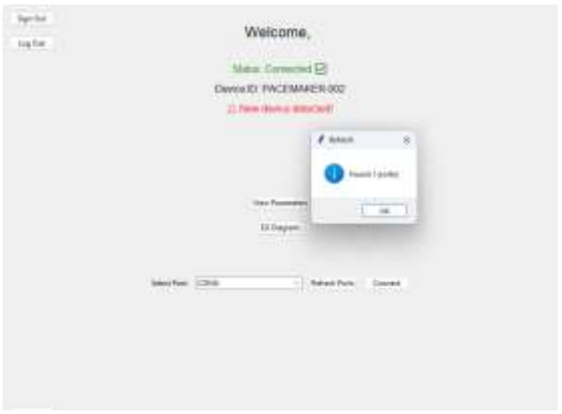
Actual Output:



Welcome,

Status: Disconnected ✗

Device ID: --



Welcome,

Status: Connected ☑

Device ID: PACEMAKER-002

New device detected!

Found 1 port(s)

Result: Pass

Trial 2 – Successful Connection / Disconnection

Purpose:

Ensure the DCM can open and close the serial connection reliably.

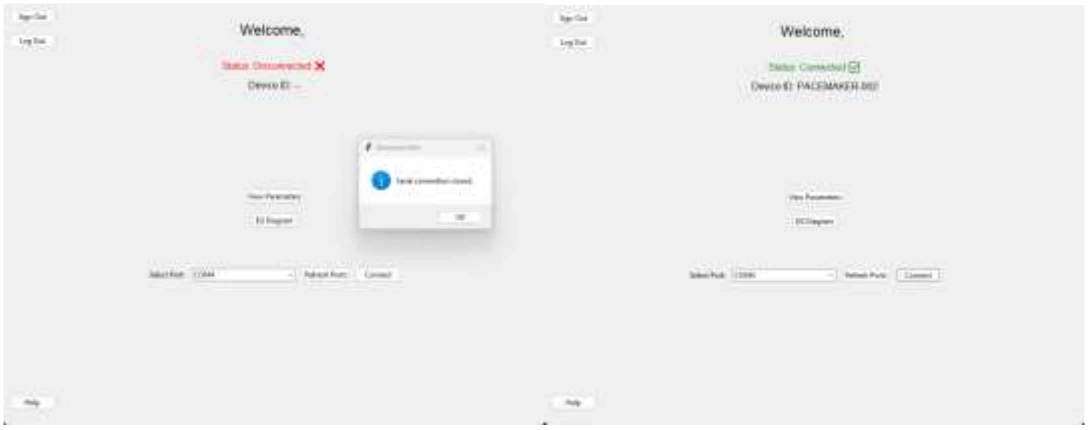
System Input:

Select a valid COM port → click connect to Connect → then click connect again to Disconnect.

Expected Output:

- The status label shows “Connected”.
- Disconnecting updates the label to “Disconnected”.
- No runtime errors occur.

Actual Output:



Result: Pass

Trial 3 – New Device Detection

Purpose:

Verify that the DCM correctly identifies when a *different* pacemaker device is connected.

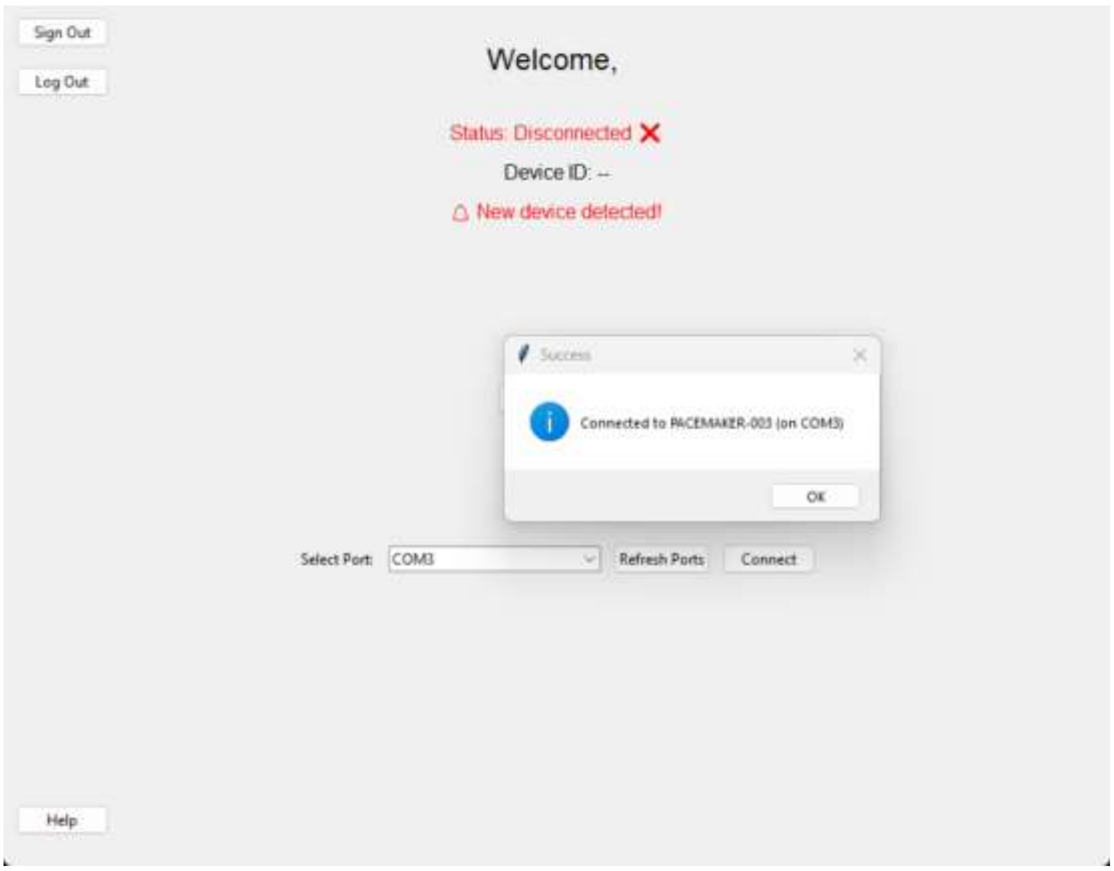
System Input:

- Stored last_connected_device = Device A
- Connect Device B

Expected Output:

- Dashboard shows the new device ID
- A red warning label “New device detected!” appears

Actual Output:



Welcome,

Status: Disconnected ✕

Device ID: --

⚠ New device detected!

Success

Connected to PACEMAKER-003 (on COM3)

Result: Pass

<u>Parameter Upload-to-Pacemaker Testing (D2)</u>

Trial 1 – Upload Blocked When Disconnected

Purpose:

Ensure that the “Load to Pacemaker” button cannot be used when no serial connection is available.

System Input:

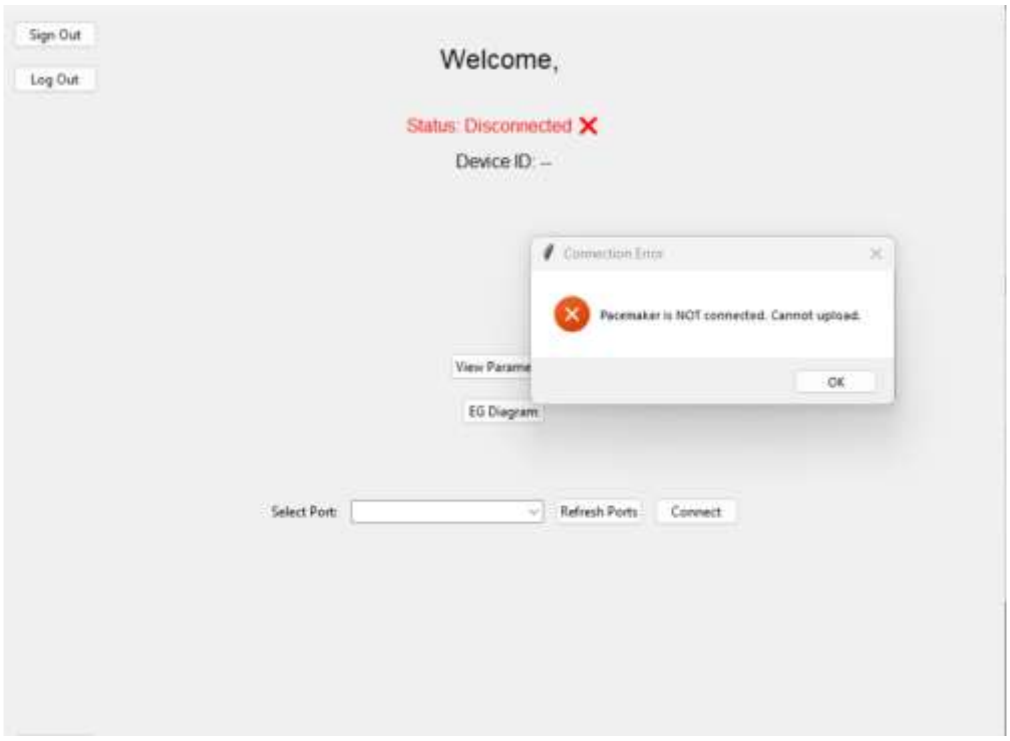
Disconnect pacemaker → click Load to Pacemaker.

Expected Output:

Error dialog appears:

“Pacemaker is NOT connected. Cannot upload.”

Actual Output:



Pacemaker is NOT connected. Cannot upload.

Result: Pass

Trial 2 – Successful Upload From JSON

Purpose:
 Verify that parameters loaded from parameters.json are encoded and transmitted to the pacemaker.

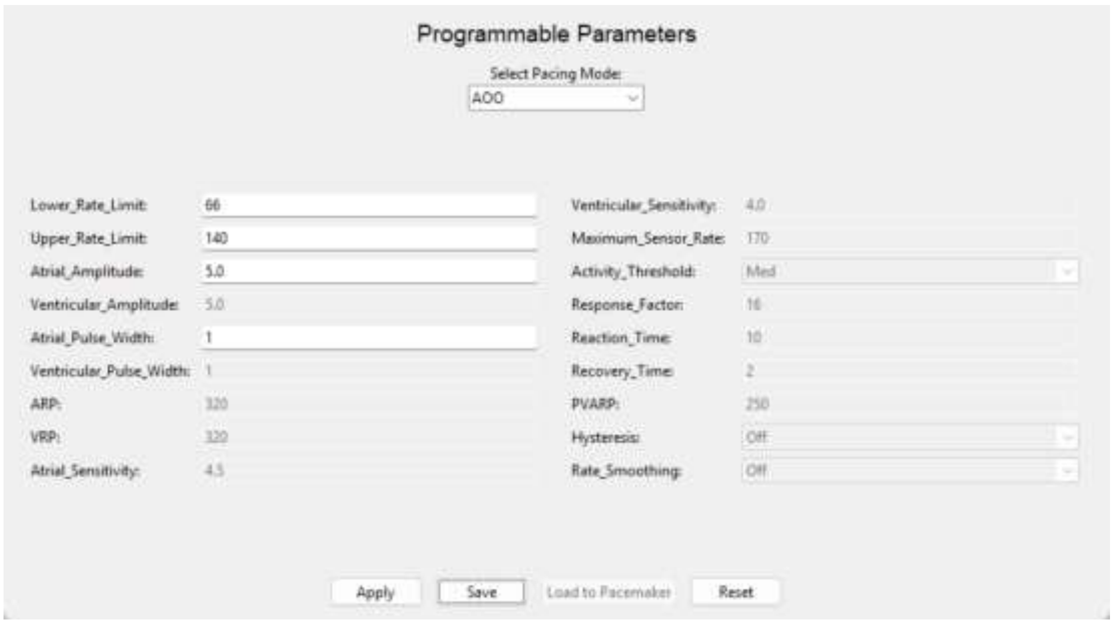
System Input:

- Connect to pacemaker
- Click Load to load JSON values
- Click Load to Pacemaker

Expected Output:

- UI fields refresh to normalized values
- Dialog "Parameters uploaded successfully" appears
- Device acknowledges the frame (green status indicator)

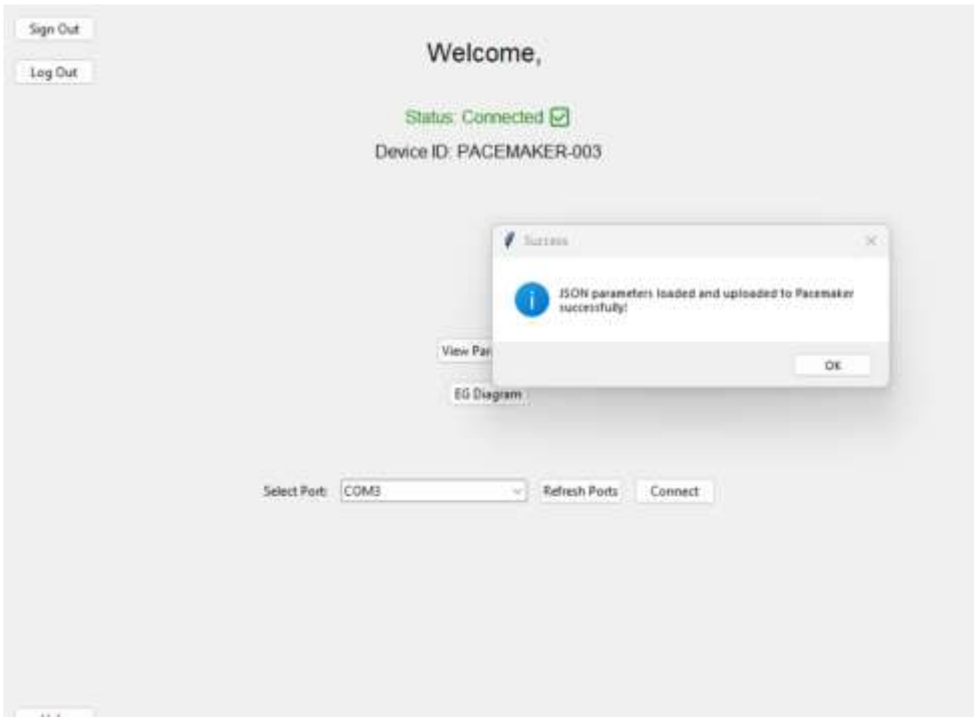
Actual Output:



Programmable Parameters

Select Pacing Mode: AOO

Lower_Rate_Limit:	66	Ventricular_Sensitivity:	4.0
Upper_Rate_Limit:	140	Maximum_Sensor_Rate:	170
Atrial_Amplitude:	5.0	Activity_Threshold:	Med
Ventricular_Amplitude:	5.0	Response_Factor:	16
Atrial_Pulse_Width:	1	Reaction_Time:	10
Ventricular_Pulse_Width:	1	Recovery_Time:	2
ARP:	320	PVARP:	250
VRP:	320	Hysteresis:	Off
Atrial_Sensitivity:	4.5	Rate_Smoothing:	Off



Welcome,

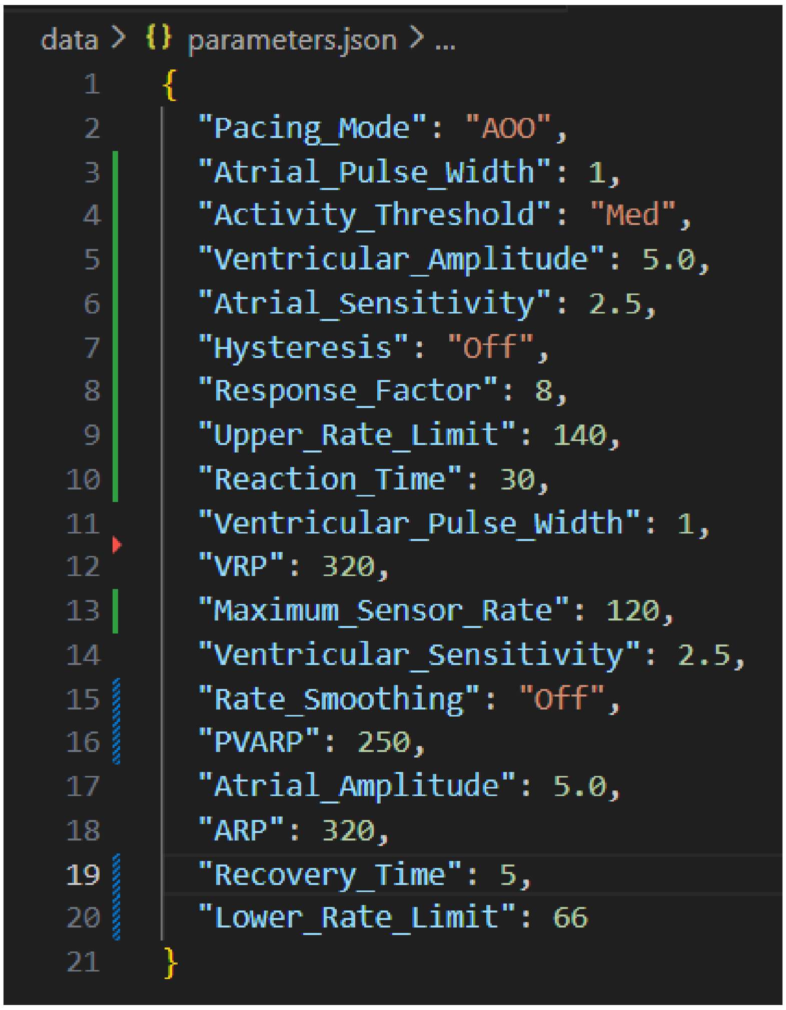
Status: Connected ✓

Device ID: PACEMAKER-003

Success

JSON parameters loaded and uploaded to Pacemaker successfully!

Select Port: COM3


data > {} parameters.json > ...

```json
{
    "Pacing_Mode": "AOO",
    "Atrial_Pulse_Width": 1,
    "Activity_Threshold": "Med",
    "Ventricular_Amplitude": 5.0,
    "Atrial_Sensitivity": 2.5,
    "Hysteresis": "Off",
    "Response_Factor": 8,
    "Upper_Rate_Limit": 140,
    "Reaction_Time": 30,
    "Ventricular_Pulse_Width": 1,
    "VRP": 320,
    "Maximum_Sensor_Rate": 120,
    "Ventricular_Sensitivity": 2.5,
    "Rate_Smoothing": "Off",
    "PVARP": 250,
    "Atrial_Amplitude": 5.0,
    "ARP": 320,
    "Recovery_Time": 5,
    "Lower_Rate_Limit": 66
}
```

Result: Pass

Trial 3 – Parameter Mismatch Guard (Before Apply)

Purpose:
Ensure that Apply checks device-stored parameters and blocks unsafe application when mismatch occurs.

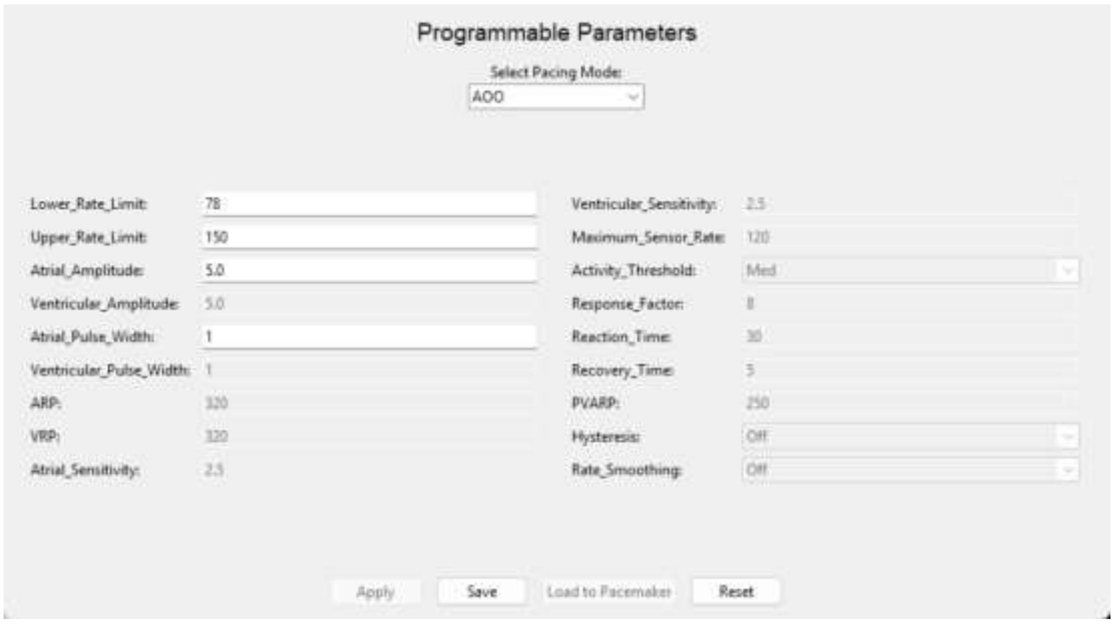
System Input:

- Modify a local parameter
- Do NOT upload to pacemaker
- Click Apply

Expected Output:
Error dialog:
"Device parameters differ from local values. Please upload first."

Apply should be aborted.

Actual Output:



Programmable Parameters

Select Pacing Mode: AOO

Lower_Rate_Limit:	78	Ventricular_Sensitivity:	2.5
Upper_Rate_Limit:	150	Maximum_Sensor_Rate:	120
Atrial_Amplitude:	5.0	Activity_Threshold:	Med
Ventricular_Amplitude:	5.0	Response_Factor:	8
Atrial_Pulse_Width:	1	Reaction_Time:	30
Ventricular_Pulse_Width:	1	Recovery_Time:	5
ARP:	320	PVARP:	250
VRP:	320	Hysteresis:	Off
Atrial_Sensitivity:	2.5	Rate_Smoothing:	Off

The apply button is grey which means that apply cannot be used before saving.

Result: Pass

Real-Time Egram Serial Testing (D2)

Trial 1 – Start Egram with Serial Data

Purpose:

Ensure Egram plotting is driven by *real serial samples* instead of D1 mock data.

System Input:

Load serial communication → Run hardware

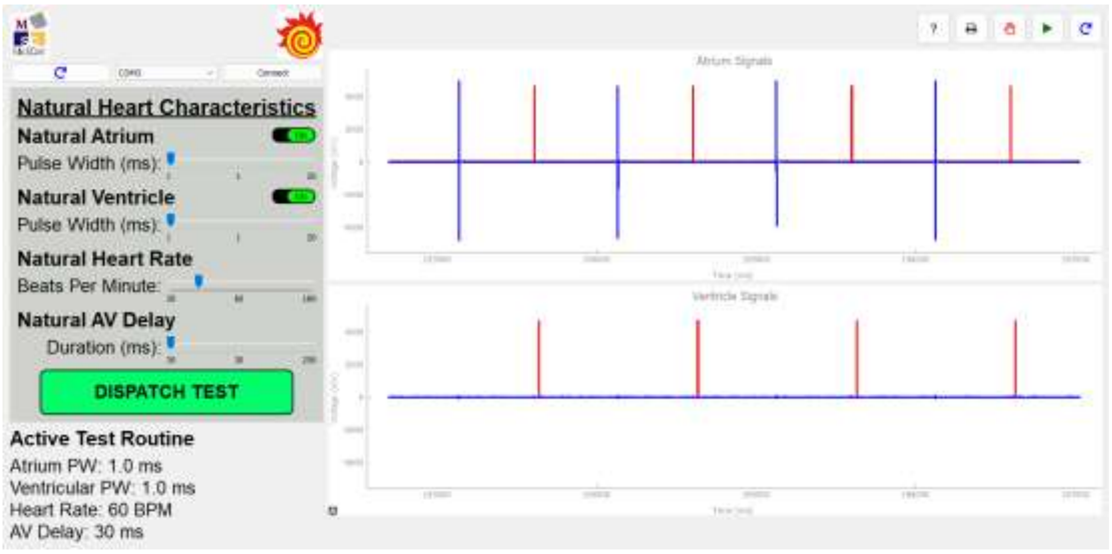
Connect → open Egram window → click Start.

Expected Output:

Continuous waveform appears for atrial/ventricular channels.

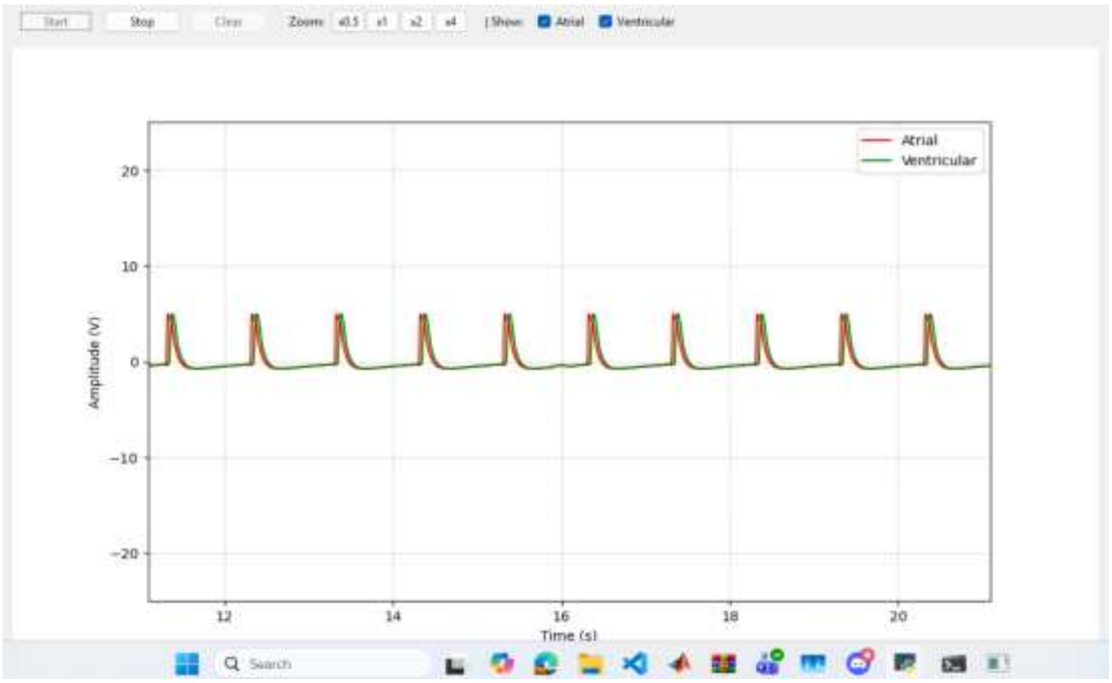
Running flag = True.

Hardware output:



Active Test Routine
Atrium PW: 1.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 60 BPM
AV Delay: 30 ms

Actual Output:



Result: Pass

3.4.6 AOO Testing

Test 1

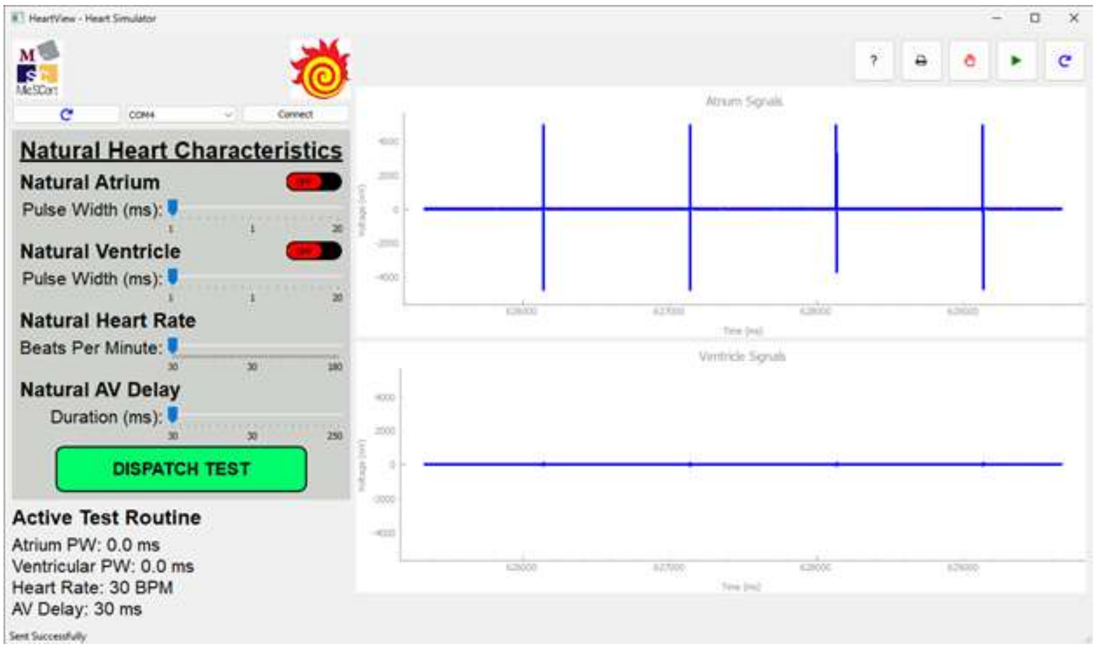
Purpose: Verify AOO delivers atrial pacing with specified parameters.

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:



Natural Heart Characteristics

Natural Atrium
Pulse Width (ms):

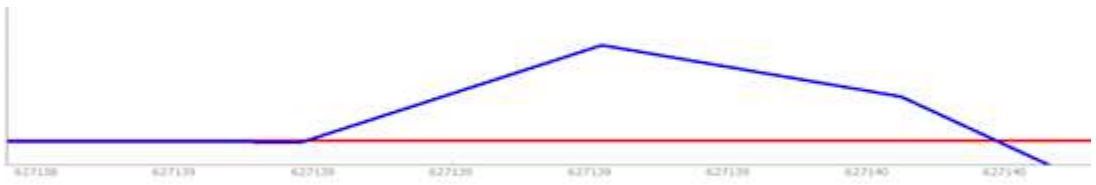
Natural Ventricle
Pulse Width (ms):

Natural Heart Rate
Beats Per Minute:

Natural AV Delay
Duration (ms):

Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 30 BPM
AV Delay: 30 ms

Sent Successfully



Result: Pass

Test 2

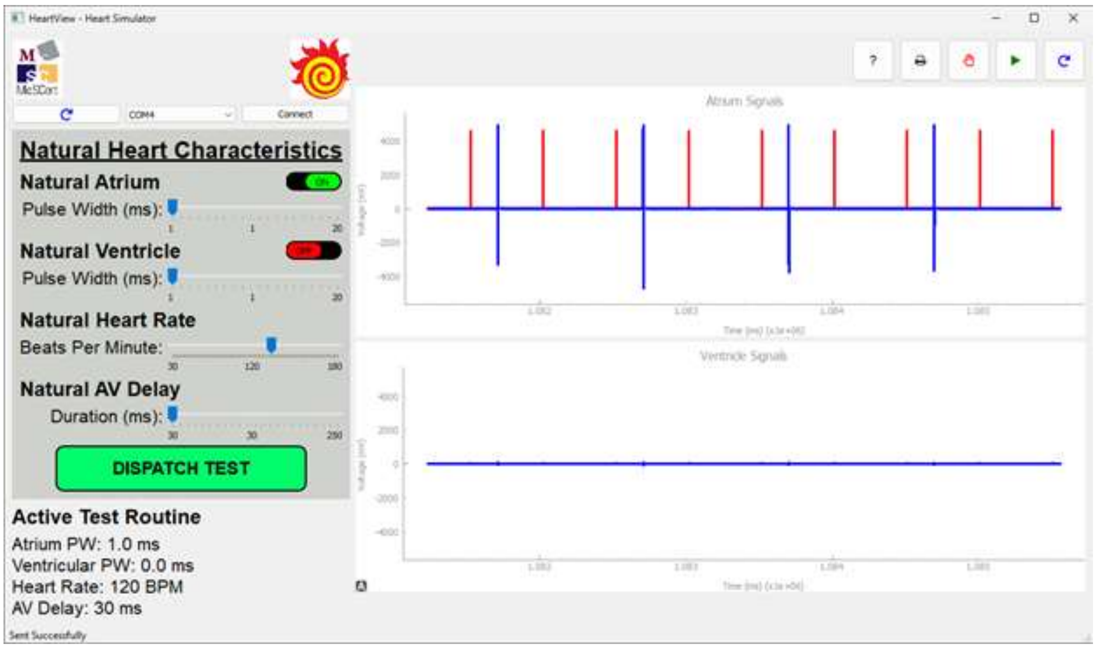
Purpose: Verify AOO continues pacing independently and ignores intrinsic atrial activity.

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:



Natural Heart Characteristics

Natural Atrium

Pulse Width (ms): 1

Natural Ventricle

Pulse Width (ms): 1

Natural Heart Rate

Beats Per Minute: 120

Natural AV Delay

Duration (ms): 30

Active Test Routine

Atrium PW: 1.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 120 BPM

AV Delay: 30 ms

Result: Pass

Test 3

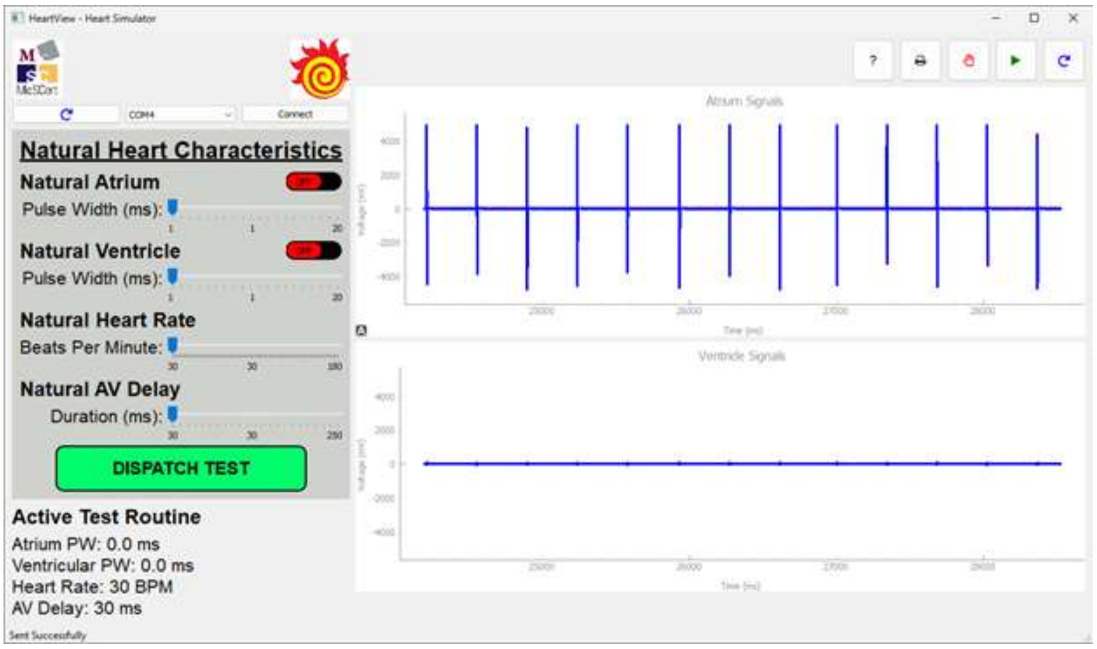
Purpose: Verify parameter validation clamps invalid inputs to safe specification limits.

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:



Natural Heart Characteristics

Natural Atrium

Pulse Width (ms):

Natural Ventricle

Pulse Width (ms):

Natural Heart Rate

Beats Per Minute:

Natural AV Delay

Duration (ms):

Active Test Routine

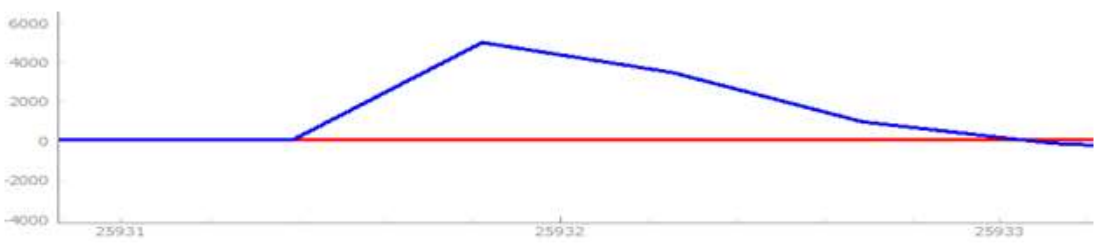
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Sent Successfully



Result: Pass

3.4.7 VOO Testing

Test 1:

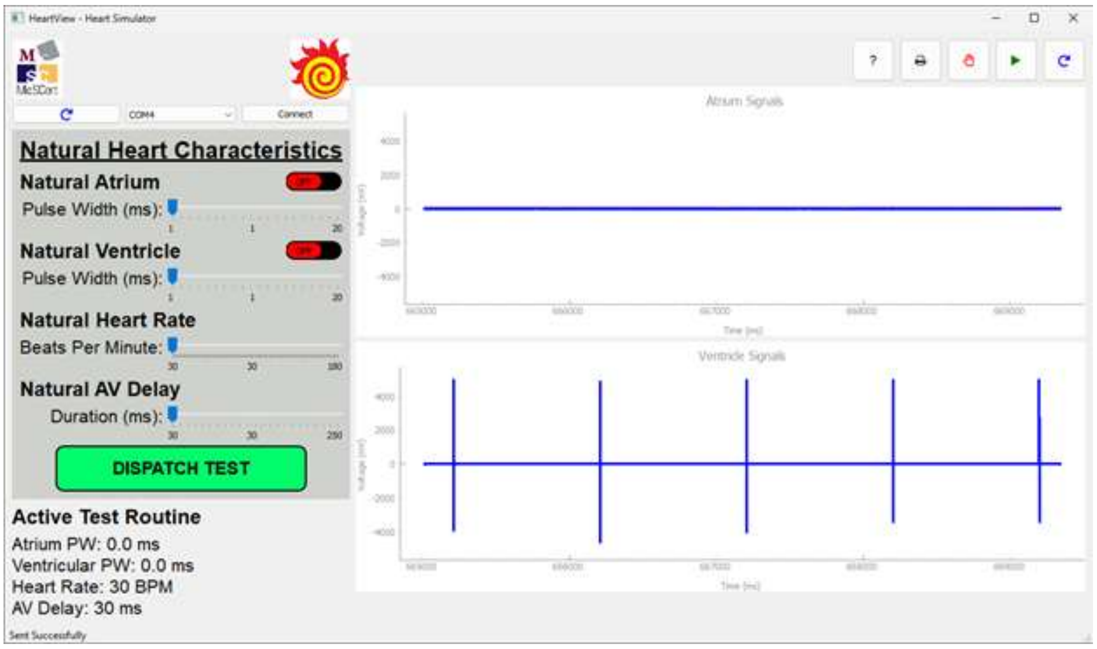
Purpose of the test: Verify VOO delivers ventricular pacing with specified parameters.

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:



Natural Heart Characteristics

Natural Atrium

Pulse Width (ms):

Natural Ventricle

Pulse Width (ms):

Natural Heart Rate

Beats Per Minute:

Natural AV Delay

Duration (ms):

Active Test Routine

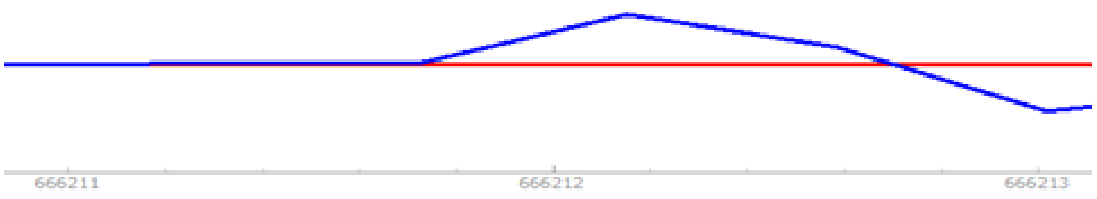
Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Sent Successfully



Result: Pass

Test 2

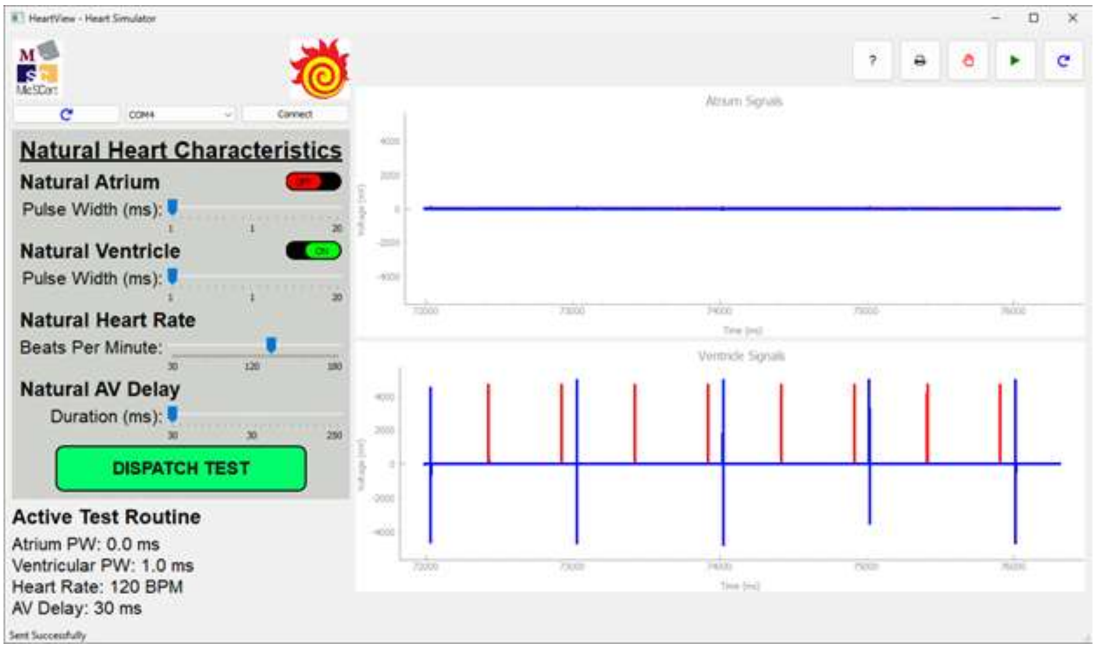
Purpose: Verify VOO continues pacing independently and ignores intrinsic ventricular activity.

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:



Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 120 BPM
AV Delay: 30 ms
Sent Successfully

Result: Pass

Test 3

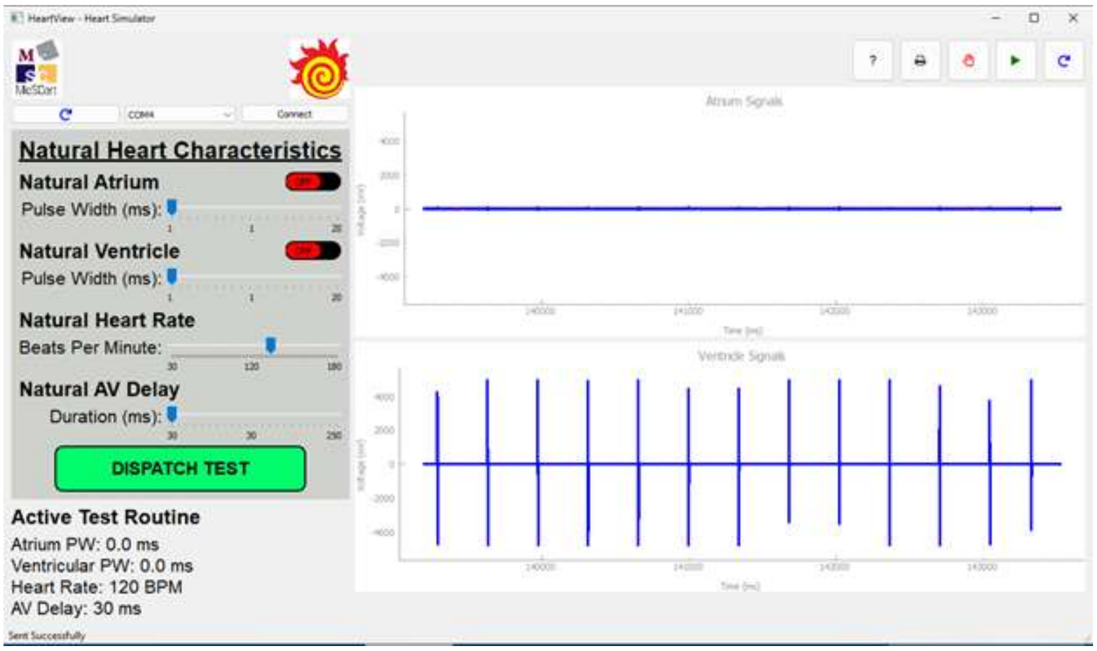
Purpose: Verify parameter validation clamps invalid inputs to safe specification limits.

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:



Natural Heart Characteristics

Natural Atrium
Pulse Width (ms):

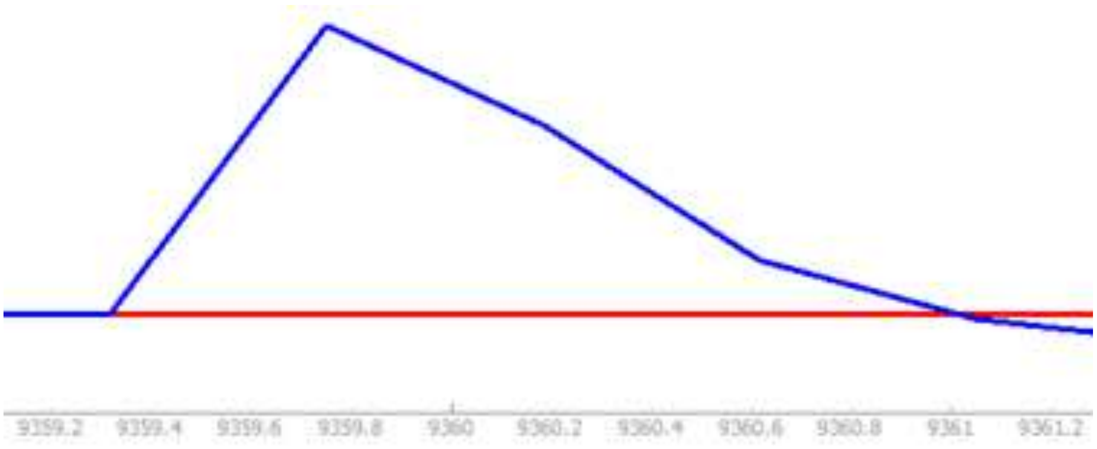
Natural Ventricle
Pulse Width (ms):

Natural Heart Rate
Beats Per Minute:

Natural AV Delay
Duration (ms):

Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 120 BPM
AV Delay: 30 ms

Sent Successfully



Result: Pass

3.4.8 AAI Testing

Test 1

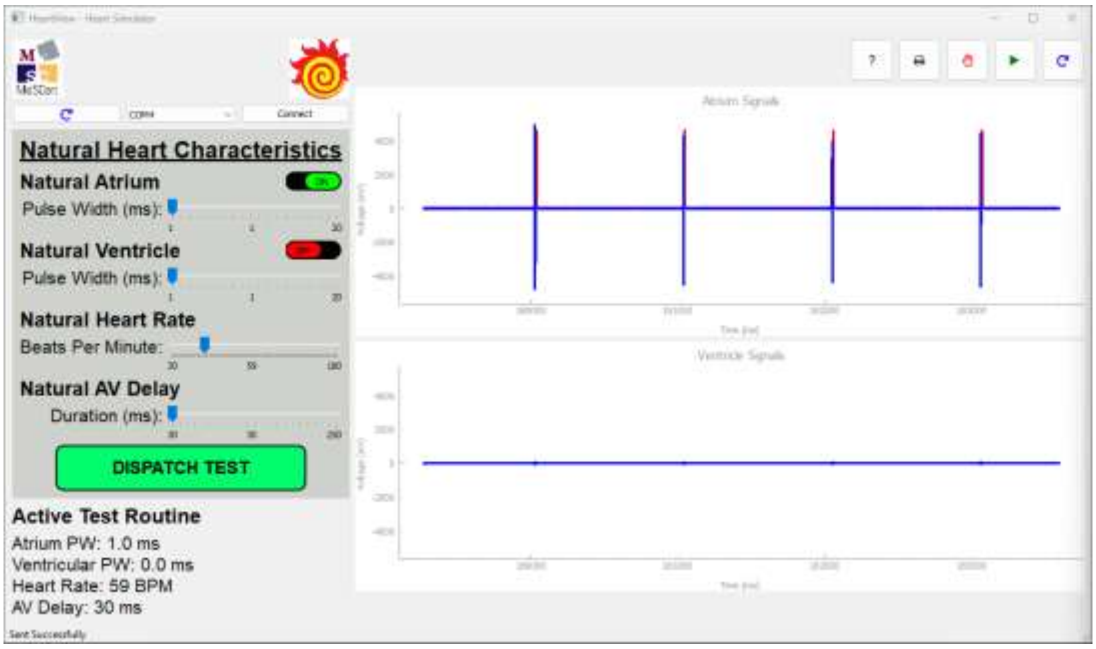
Purpose: Verify AAI mode delivers atrial pacing when intrinsic atrial activity falls below LRL.

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:



Active Test Routine
Atrium PW: 1.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 59 BPM
AV Delay: 30 ms
Sent Successfully

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

Result: Pass

Test 2

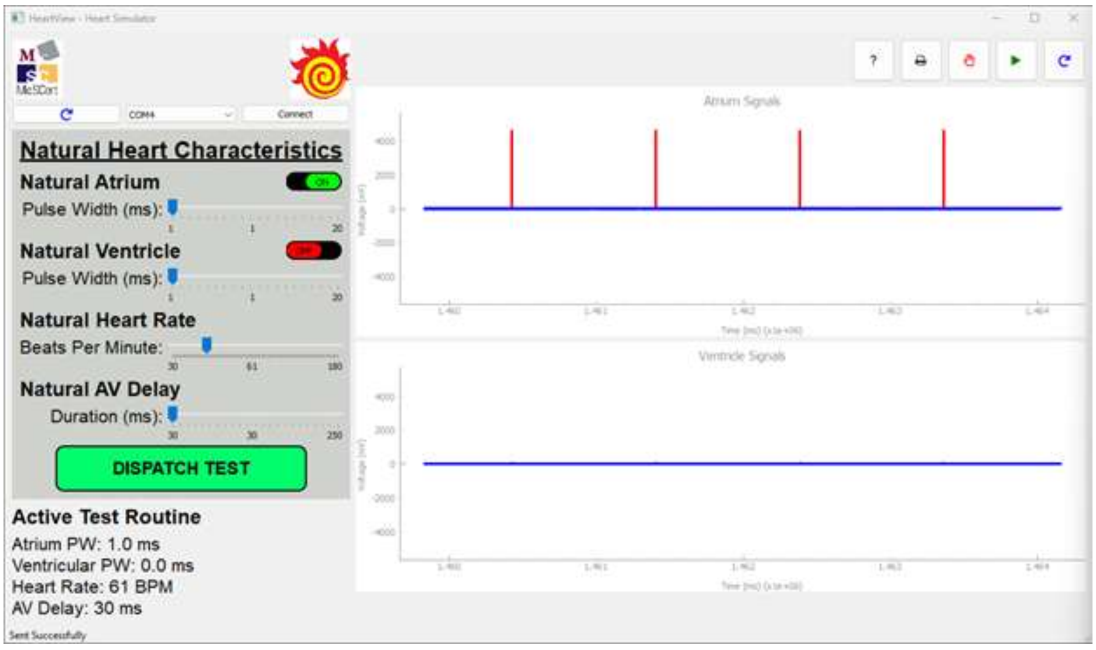
Purpose: Verify AAI mode does not deliver atrial pacing when intrinsic atrial activity exceeds LRL.

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:



Natural Heart Characteristics

Natural Atrium
Pulse Width (ms): 1

Natural Ventricle
Pulse Width (ms): 1

Natural Heart Rate
Beats Per Minute: 61

Natural AV Delay
Duration (ms): 30

Active Test Routine
Atrium PW: 1.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 61 BPM
AV Delay: 30 ms

Sent Successfully

Result: Pass

Test 3

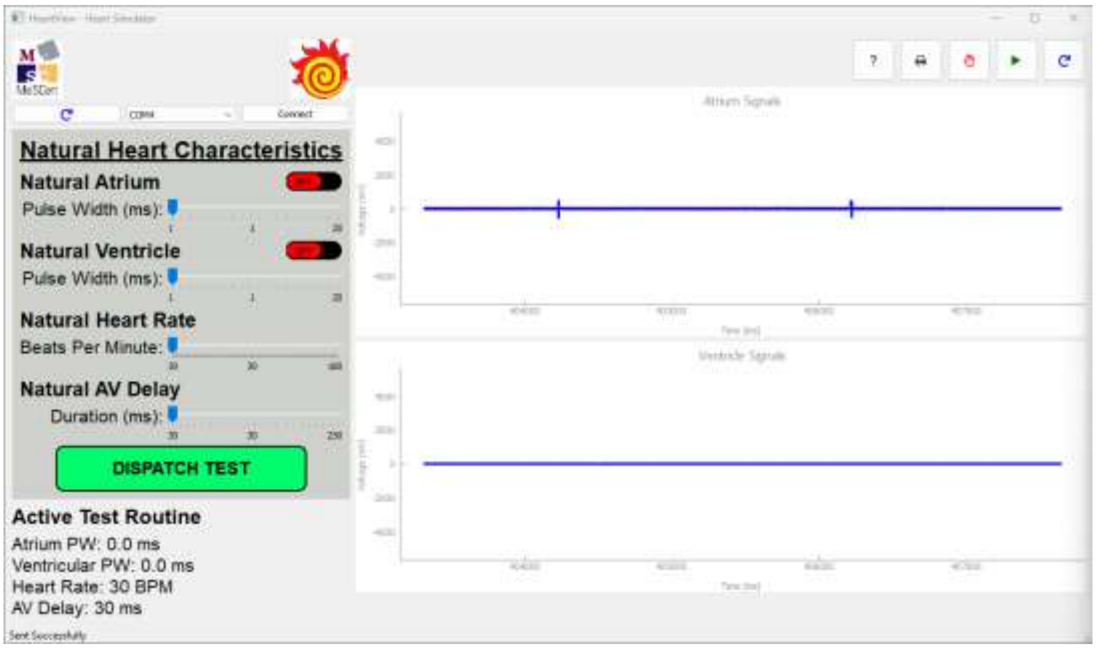
Purpose: Verify AAI mode delivers pacing at the Lower Rate Limit when no natural atrial activity is detected and validate parameter clamping to specification limits.

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:



Natural Heart Characteristics

Natural Atrium

Pulse Width (ms):

Natural Ventricle

Pulse Width (ms):

Natural Heart Rate

Beats Per Minute:

Natural AV Delay

Duration (ms):

Active Test Routine

Atrium PW: 0.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 30 BPM

AV Delay: 30 ms

Sent Successfully

Result: Pass

3.4.9 VVI Testing

<u>Test 1</u>

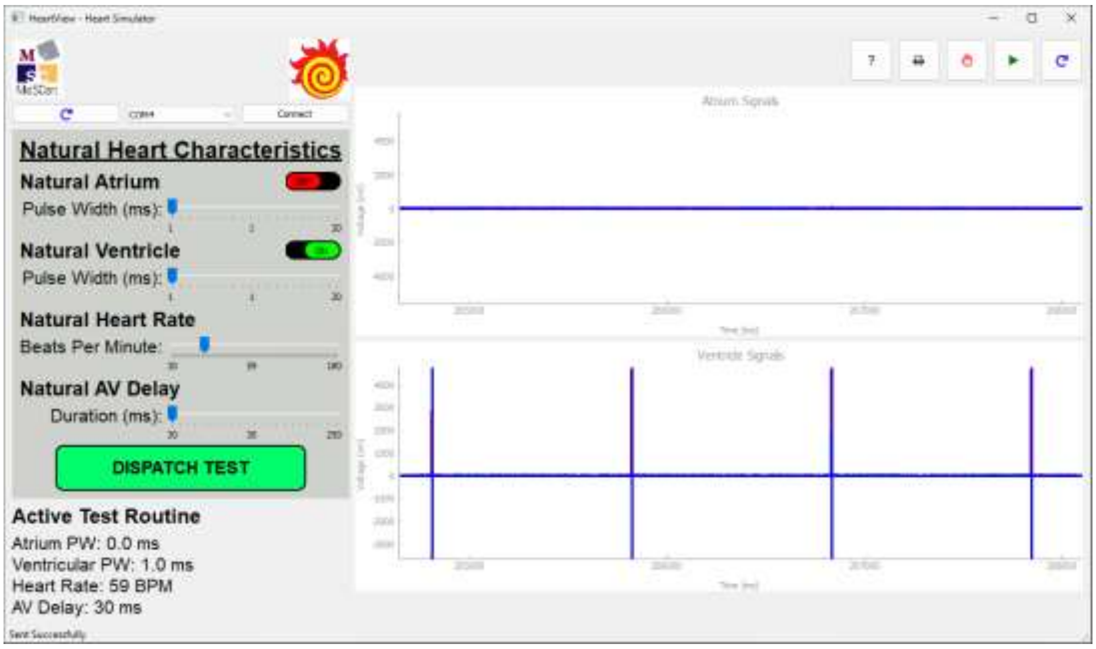
Purpose: Verify VVI mode delivers ventricular pacing when intrinsic ventricular activity falls below LRL.

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:



Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 59 BPM
AV Delay: 30 ms
Sent Successfully

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

Result: Pass

Test 2

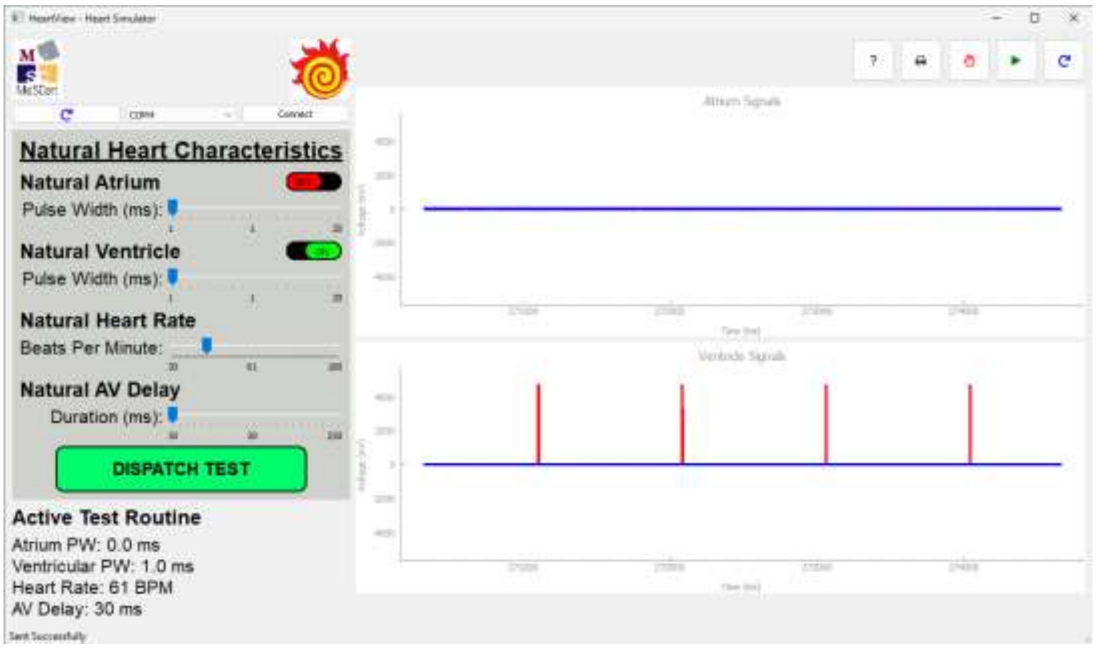
Purpose: Verify VVI mode does not deliver ventricular pacing when intrinsic ventricular activity exceeds LRL.

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:



Natural Heart Characteristics

Natural Atrium

Pulse Width (ms):

Natural Ventricle

Pulse Width (ms):

Natural Heart Rate

Beats Per Minute:

Natural AV Delay

Duration (ms):

Active Test Routine

Atrium PW: 0.0 ms

Ventricular PW: 1.0 ms

Heart Rate: 61 BPM

AV Delay: 30 ms

Sent Successfully

Result: Pass

<u>Test 3</u>

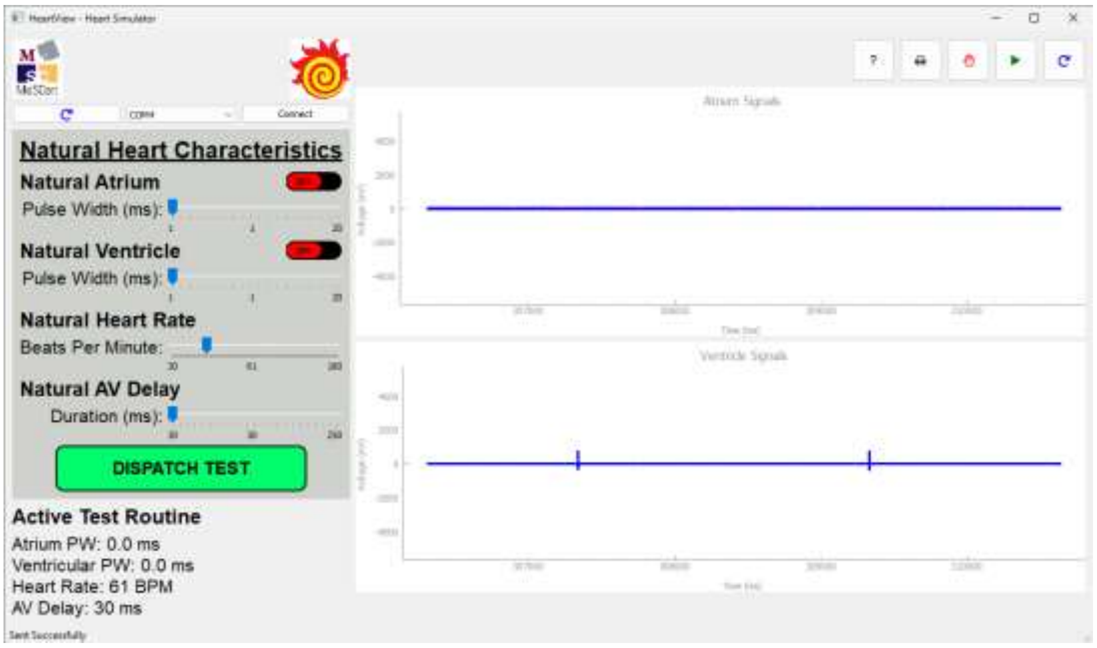
Purpose: Verify VVI mode delivers pacing at the Lower Rate Limit when no natural ventricular activity is detected and validate parameter clamping to specification limits.

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:



Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 61 BPM
AV Delay: 30 ms
Sent Successfully

Result: Pass

3.4.12 Rate Adaptation Testing

During the demo, all four rate-adaptive modes (AOOR, VOOR, AAIR, VVIR) behaved like non-rate-adaptive modes. The pacing rate stayed at a fixed multiple of the Lower Rate Limit or at some constant value and never increased with movement. We tried to debug this by adding scopes to watch the accelerometer signal in Simulink, but after a crash the model started getting stuck at 95% loading and we could not reliably test on hardware.

The most likely reasons are:

- The accelerometer was not providing usable data. This could be due to incorrect initialization, wiring issues, or a communication problem on the sensor bus, so the activity signal going into the rate adaptation block was effectively always zero.
- The Simulink configuration around the accelerometer was wrong. Examples include an incorrect device address, wrong data type, or incorrect scaling from raw acceleration into the 0-5 activity scale. In that case, the computed activity value would never cross the Activity Threshold, even when the device was shaken.
- Timing and integration issues prevented the accelerometer data from updating properly. After adding scopes, serial communication, and the rate-adaptive logic, sample-time mismatches, execution-order problems, or CPU overload could cause the accelerometer block to run out of sync with the rest of the model, leaving the activity signal stuck or very slow to change. Misconfigured sampling times, missing triggered subsystems for the accelerometer, or data-type mistakes (such as unintended integer division) can also shift the effective timing by an integer multiple and disturb both pacing and serial communication.

Because of these issues, the rate-adaptive behavior in our implementation could not be trusted. Accelerometer values appeared incorrect, the internal activity level and adapted rate could not be confirmed, and the model sometimes failed to fully load onto the hardware. As a result, we did not perform or claim any real hardware testing for AOOR, VOOR, AAIR, or VVIR in this deliverable. Instead, for completeness, we include a set of hypothetical test cases that describe how these modes would be validated (using controlled accelerometer inputs, activity thresholds, reaction/recovery times, and HeartView scenarios) if the accelerometer interface and rate-adaptive pipeline were working reliably.

3.4.13 AOOR Testing

Test 1

Purpose: Verify AOOR increases pacing rate above LRL when sustained activity exceeds the Activity Threshold while staying $\leq$ MSR.

Input conditions:

- Amplitude: 3.5 V
- Sensitivity: 8 mV
- LRL: 60 bpm
- MSR: 120 bpm
- Activity Threshold: medium “walking” level
- Response Factor: 8
- Reaction Time: 30 s
- Recovery Time: 5 min
- Mode: 4 (AOOR)

Expected output: After approximately 30 s of above-threshold activity, the paced atrial interval shortens from about 1000 ms toward about 500 ms (rate between 60 and 120 bpm), with no interval shorter than the MSR limit.

3.4.14 VOOR Testing

Test 1

Purpose: Verify VOOR increases ventricular pacing rate above LRL when sustained activity exceeds the Activity Threshold while staying $\leq$ MSR

Input conditions:

- Amplitude: 3.5 V
- Sensitivity: 8 mV
- LRL: 70 bpm
- MSR: 130 bpm
- Activity Threshold: medium "walking/jogging" level
- Response Factor: 8
- Reaction Time: 30 s
- Recovery Time: 5 min
- Mode: 5 (VOOR)

Expected output: After approximately 30 s of above-threshold activity, the paced ventricular interval shortens from about 857 ms toward about 462 ms (rate between 70 and 130 bpm), with no interval shorter than the MSR limit.

3.4.15 AAIR Testing

Test 1

Purpose: Verify AAIR behaves like AAI demand pacing but uses the adapted rate, inhibiting pacing when intrinsic atrial rate exceeds the adapted rate.

Input conditions:

- Amplitude: 3.5 V
- Sensitivity: 2 mV
- LRL: 60 bpm
- MSR: 120 bpm
- Activity Threshold: medium level
- Response Factor: 8
- Reaction Time: 30 s
- Recovery Time: 5 min
- ARP: 250 ms
- Intrinsic atrial rate: 90 bpm
- Mode: 6 (AAIR)

Expected output: After activity raises the adapted rate above 60 bpm toward roughly 90 bpm, intrinsic atrial beats at 90 bpm inhibit pacing whenever they occur before the adapted timeout; atrial pacing only occurs when no intrinsic beat is sensed within the adapted interval, with ARP enforced and no pacing faster than MSR.

3.4.16 VVIR Testing

<u>Test 1</u>

Purpose: Verify VVIR behaves like VVI demand pacing but uses the adapted rate, inhibiting pacing when intrinsic ventricular rate exceeds the adapted rate.

Input conditions:

- Amplitude: 3.5 V
- Sensitivity: 2 mV
- LRL: 60 bpm
- MSR: 130 bpm
- Activity Threshold: medium level
- Response Factor: 8
- Reaction Time: 30 s
- Recovery Time: 5 min
- VRP: 250 ms
- Intrinsic ventricular rate: 90 bpm
- Mode: 7 (VVIR)

Expected output: Expected output: After activity raises the adapted rate above 60 bpm toward roughly 90 bpm, intrinsic ventricular beats at 90 bpm inhibit pacing whenever they occur before the adapted timeout; ventricular pacing only occurs when no intrinsic beat is sensed within the adapted interval, with VRP enforced and no pacing faster than MSR.

4. Assurance

DCM Assurance Case. This forms an assurance argument based on the requirements for Deliverable 2, stating that the Device Controller-Monitor (DCM) is safe in its operation, consistent and reliable. This DCM sets up configuration of programmed parameters, data validation on what the user inputs, sends values to the pacemaker, validates data of stored values, and gives actual Egrams instantly. The assertion of an assurance case includes demonstrating that the DCM avoids potentially dangerous parameter configurations, inconsistent device states, and reliable communication to the pacemaker. This forms an assurance argument based on the requirements for Deliverable 2, stating that the Simulink pacemaker model is safe in its operation, reliable, and deterministic.

4.1 Top Level Safety Claim.

C0 − The DCM ensures that it ensures safe, valid, and correctly used programmable control parameters in the physical device of the pacemaker. Subclaims pertaining to UI validation, transmission accuracy, device–DCM consistency, and safe visualization of real-time signals are then compiled over this high-level claim.

C0 − The Simulink pacemaker model ensures safe and physiologically appropriate cardiac pacing across all modes while maintaining timing accuracy and preventing hazardous rate or amplitude conditions.

4.2 Strategy.

• S1: Show that the DCM prevents unsafe or invalid parameter values by strict validation.

• S2: Show that the DCM will guarantee that the parameters transmitted during serial connection are complete, well-formed, and acknowledged by the device.

• S3: Show that the DCM checks the pacemaker-stored parameters and prevents inappropriate parameters from being applied.

• S4: Proves the DCM can consume real-time egram data such as disconnects and unstable signals without causing problems.

• S5: Show that the DCM UI prevents wrong action, incomplete workflow, or status ambiguity. All of those approaches have specific subclaims and evidence to support them.

SIMULINK:

S1 — Parameter Validation and Saturation

Claim C1: The Input Hardware Hiding module ensures that all programmable parameters are constrained to their safe operating ranges before being passed to the Main module.

Justification / Argument:

- All input parameters (LRL, URL, amplitudes, pulse widths, refractory periods, MSR, activity threshold, response factor, reaction time, recovery time) are processed through dedicated saturation blocks.
- Saturation limits match the specified ranges in the PACEMAKER requirements document (e.g., LRL: 30–175 BPM, URL: 50–180 BPM, amplitudes: 0–7.0V excluding 3.3V gap).
- Unit conversions (BPM to ms, voltage to PWM duty cycle) are performed consistently using validated formulas.
- Accelerometer data is filtered and bounded before computing adapted rate.
- Invalid or out-of-range values are automatically clamped to nearest safe boundary.

- S1: Show that the Input Hardware Hiding module enforces parameter bounds through saturation and prevents invalid values from reaching the control logic.

- S2: Show that all pacing modes execute deterministically with verified timing accuracy and refractory period compliance.

- S3: Show that rate adaptation logic maintains physiological constraints (LRL ≤ adapted_rate ≤ MSR) and smooth transitiond during the recovery time.

4.3 Subclaims and evidence.

S1 — Validation to ensure parameter safety.

Claim C1: The DCM ensures that all clinician-entered parameters are valid, within specification, step-rounded, and cross-checked before being stored or transmitted.

Justification / Argument:

• The parameter input is validated in the centralized ParamEnum layer.

• Acceptance is pre-checked with step rounding, range limits, and special-case rules (e.g., amplitude gaps, LRL ≤ URL).

• Invalid inputs result in blocking error dialogs; no unsafe values can exit the UI.

• LRL/URL violation error dialogs.

• Amplitude gap error dialogs (3.3V invalid range).

• UI rejection of malformed, empty, or out-of-range data.

S2 – Correctly encoded parameters and transmission.

Claim C2: The DCM guarantees that all the specified parameters transmitted to the pacemaker are encoded correctly within the required 29-byte frame.

Justification / Argument:

• All upload parameters are managed centrally by PacemakerCommunication.upload_parameters().

• Values are deterministically encoded using scaling rules (e.g., amplitude × 100).

• Transmission is blocked if the connection is not active.

• Device returns acknowledgment verifying acceptance.

Evidence:

• Serial communication testing trial 1–2.

• Parameter Upload Testing trial 1–2.

• Error dialog: “Pacemaker not connected. Cannot upload.”

• Frame from device receiving acknowledgment.

S3 – Consistency Between DCM and Pacemaker.

Claim C3: The DCM checks that the parameters on the pacemaker match the doctor’s requested configuration.

Justification / Argument:

• DCM requests device-stored parameters after uploading.

• Values are compared against local normalized values.

• If there is a mismatch, the Apply is blocked and the user is instructed to reload.

• Avoids silent divergence between UI settings and device behavior.

Evidence:

• Parameter Upload Testing trial 3: mismatch guard.

• UI warning dialog: “Device parameters differ from local values.”

• Confirmed correct behavior when manipulating device-side values by design.

S4 - Handling real-time Egram messages safely.

The DCM handles real-time atrial/ventricular electrograms and avoids unsafe UI states during disconnects or unstable communication.

Justification / Argument:

• EgramWindow only starts sampling once the connection is stable.

• Upon disconnection, noise, or unstable signal, sampling stops automatically.

• UI prevents conflicting actions (no double start; clear disabled when running).

• Stale, misleading, or corrupted clinical data cannot be displayed.

Evidence:

• Egram Serial Testing trials 1–3.

• Auto-stop on disconnect.

• Noise/unstable state detection.

• Running flag lockout logic.

S5 — Safe & Deterministic UI behavior.

Claim C5: The DCM avoids unsafe and ambiguous workflows with strong UI guards.

Justification / Argument:

• "Load to Pacemaker" disabled when disconnected.

• Apply only allowed after Save → Load → Verify sequence.

• Single-instance windows (no duplicate or concurrent sessions by accident).

• Device identity check prevents inadvertent programming of the wrong device.

• Dashboard clearly shows connection + device ID + new device warning.

Evidence:

• Serial Communication Testing trial 3: detecting new devices.

• Dashboard UI tests on connection state transitions.

• ParameterWindow button enables/disable logic.

• Window management in DashboardWindow (single-instance lifts).

SIMULINK:

S1 — Parameter Validation and Saturation

Claim C1: The Input Hardware Hiding module ensures that all programmable parameters are constrained to their safe operating ranges before being passed to the Main module.

Justification / Argument:

- All input parameters (LRL, URL, amplitudes, pulse widths, refractory periods, MSR, activity threshold, response factor, reaction time, recovery time) are processed through dedicated saturation blocks.
- Saturation limits match the specified ranges in the PACEMAKER requirements document (e.g., LRL: 30–175 BPM, URL: 50–180 BPM, amplitudes: 0–7.0V excluding 3.3V gap).
- Unit conversions (BPM to ms, voltage to PWM duty cycle) are performed consistently using validated formulas.
- Accelerometer data is filtered and bounded before computing adapted rate.
- Invalid or out-of-range values are automatically clamped to nearest safe boundary.

S2 — Deterministic Timing and Refractory Period Compliance

Claim C2: All pacing modes execute with deterministic timing accuracy, ensuring correct pacing intervals, pulse widths, and refractory periods are honored without timing drift or race conditions.

Justification / Argument:

- Simulink model uses fixed-step solver with 1 ms time step, ensuring deterministic execution.
- All Stateflow transitions use after(duration, msec) with validated timing calculations.
- Charging and pacing states follow strict sequencing: Idle → Charging → Pacing → Idle (or Sensing in demand modes).
- Refractory periods (ARP for atrial, VRP for ventricular) are enforced in AAI, VVI, AAIR, VVIR modes to prevent inappropriate sensing during blanking windows.
- Timing calculations avoid integer overflow by using double data types throughout.
- Rate adaptation does not violate timing constraints (adapted_rate_ms always ≥ pulse_width_ms).

4.4 Conclusion

The assurance case depicts that the DCM is an effective, reliable and safe, and verifiable communication intermediary between clinician and pacemaker device. With robust parameter validation, deterministic protocol encoding, device-DCM consistency, reliable live egram handling, and protected UI workflows, the DCM minimizes potential for misconfiguration, communication breakdown, or unsafe device operation. Accordingly, the top-level claim is confirmed as:

C0-Each of the programmable parameters for pacemakers is safe, valid, and correctly applied by the DCM on the physical device.

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.