Deliverable 2: Documentation

Group #22

Fall 2025

Table of Contents

1 Group Members 1

2 Introduction 1

3 Pacemaker 1

3.1 Requirements 1

3.2 Mode Specific Requirements 2

3.2.1 AOO 2

3.2.2 VOO 2

3.2.3 AAI 2

3.2.4 VVI 2

3.2.5 AOOR 3

3.2.6 VOOR 3

3.2.7 AAIR 3

3.2.8 VVIR 3

3.3 Design 3

3.3.1 System Architecture 3

3.3.2 Programmable Parameters 5

3.3.3 Finite State Machine Design 6

3.4 Requirements Potential Changes 13

3.5 Design Decision Potential Changes 13

3.6 Module Description 14

3.6.1 Sensing Subsystem 14

3.6.2 Parameters Subsystem 14

3.6.3 Pacemaker Stateflow Subsystem 15

3.6.4 Hardware Hiding for Outputs Subsystem 16

3.6.5 Adaptive Pacing Control Subsystem 17

3.6.6 Serial Receive Subsystem 20

3.6.7 Serial Transmit Subsystem 22

3.7 Testing 22

4 DCM 30

4.1 Requirements 30
4.2 Design 31
4.2.1 Software Structure 31
4.2.2 Component Interaction/UI Workflow 31
4.2.3 Design-to-Requirements 32
4.3 Requirements Potential Changes 36
4.4 Design Decision Potential Changes 36
4.5 Module Description 37
4.6 Testing 41
5 Assurance Case 48
5.1 Requirements-Focused Safety Rationale 48
5.2 Hazard Identification 48
5.3 Hazard Mitigation Strategies 49
5.3.1 Failure to Pace 49
5.3.2 Oversensing 49
5.3.3 Under-sensing 49
5.3.4 Over/Under Pacing 50
5.3.5 Wrong Chamber Pacing 50
5.4 Mode-Specific Safety Requirements 50
5.4.1 AOO Safety 50
5.4.2 VOO Safety 50
5.4.3 AAI Safety 50
5.4.4 VVI Safety 50
5.4.5 AOOR Safety 51
5.4.6 VOOR Safety 51
5.4.7 AAIR Safety 51
5.4.8 VVIR Safety 51
6 GenAI Usage 51

List of Figures

Figure 1: Simulink System Architecture 4
Figure 2: AOO Simulink State 7
Figure 3: VOO Simulink State 8
Figure 4: AAI Simulink State 9
Figure 5: VVI Simulink State 10
Figure 6: AOOR Simulink State 10
Figure 7: VOOR Simulink State 11
Figure 8: AAIR Simulink State 12
Figure 9: VVIR Simulink State 13
Figure 10: Sensing Subsystem 14
Figure 11: Programmable Parameters Subsystem 15
Figure 12: Pacemaker Stateflow Subsystem 16
Figure 13: Hardware Hiding Subsystem 17
Figure 14: Rate Adaptive Subsystem 18
Figure 15: Rate Adaptive Pacing Subsystem - Smoothing Function 19
Figure 16: Rate Adaptive Pacing Subsystem - Linear Rate Mapping 19
Figure 17: Rate Adaptive Pacing Subsystem – Time Rate Transformation FSM for Gradual Increase/Decrease 20
Figure 18: Serial Receive Subsystem 21
Figure 19: Initialize and Set Parameters FSM 21
Figure 20: Serial Transmit Subsystem 22
Figure 21: Heartview Output for AOO Test 23
Figure 22: Heartview Output for VOO Test 1 24
Figure 23: Heartview Output for AAI Test 1 25
Figure 24: Heartview Output for AAI Test 2 26
Figure 25: Heartview Output for VVI Test 1 27
Figure 26: Heartview Output for VVI Test 2 28
Figure 27: Heartview Output for AOOR Test 29
Figure 28: Heartview Output for VOOR Test 30
Figure 29: Authentication. The DCM enforces a local maximum of 10 users for D1; a successful login routes to the Dashboard. 33
Figure 30: Dashboard (Connected). Connection status, current mode, battery level, and static device info are visible post-login; navigation to Modes/Parameters is available. 34
Figure 31: Pacemaker Connection Modal. Users select a COM port and test the (simulated) connection; success toggles the global status to Online. 34
Figure 32: Pacing Modes. All Deliverable 1 modes are presented with chamber/sensing/response; the active mode is highlighted and re-apply is disabled. 35

Figure 33: Programmable Parameters. LRL, URL, AA, APW, VA, VPW, VRP, ARP are adjustable via slider, increment/decrement arrows, or direct numeric input; the summary updates live..... 36
Figure 34: Basic Registration Test Output..... 41
Figure 35: Dashboard Display Test Output..... 42
Figure 36: Dashboard Offline Test Output..... 43
Figure 37: Pacemaker Connection Test Output..... 44
Figure 38: Pacing Modes Visibility Test Output..... 45
Figure 39: Parameters Page Visibility Test Output..... 46

List of Tables

Table 1: Parameters Used for Each Bradycardia Therapy Mode..... 5
Table 2: Mapping between Technical and Corresponding Pin Names specific to K64F Microcontroller..... 5
Table 3: Pacemaker Modes and their Respective Configuration Value..... 6
Table 4: State Transition Table for AOO..... 7
Table 5: State Transition Table for AAI..... 9
Table 6: State Transition Table for AOOR..... 11
Table 7: State Transition Table for AAIR..... 12
Table 8: AOO Test 1..... 23
Table 9: VOO Test 1..... 23
Table 10: AAI Test 1..... 24
Table 11: AAI Test 2..... 25
Table 12: VVI Test 1..... 26
Table 13: VVI Test 2..... 27
Table 14: AOOR Test..... 28
Table 15: VOOR Test..... 29
Table 16: Descriptions of Each Module in DCM..... 37
Table 17: FMEA Hazard Analysis..... 49

1 Group Members

Name	Student Number	MacID
Avinash Cheeranjie	400530770	cheerana
Habel Kingson	400528728	kingsonh
Abdullah Abusalout	400534459	abusaloa
Marwan Ali	400450021	ali186
Ratish Gupta	400523218	guptar76
Ziad Elmaddah	400529599	elmaddaz

2 Introduction

This document highlights the extensive process involved with developing an effective pacemaker system, while also ensuring functionality as a safety-critical system. The pacemaker system's purpose is to ensure that patient health and safety are up-held when providing life-critical therapy through documented, verified, and validated software. More specifically, this document outlines two key components to the pacemaker system, the Simulink State Flows, as well as the DCM (Device Controller Monitor). The Simulink State Flows model eight essential pace-making modes which provide bradycardia therapy to a simulated heart, through safe control/regulation of its responses. In addition to this, the DCM provides a graphical user interface that allows for interaction and adjustments to be made with the pacemaker, through enabling a serial communication protocol. Overall, these two key systems conjoin together to ensure that clinicians can effectively advise patients on pacing abnormalities, to provide them with a better quality of life.

3 Pacemaker

3.1 Requirements

The following system requirements govern the overall Simulink design:

- Programmable Parameters**:** The system should allow the setting of all the pacemaker parameters through UART serial communication with the DCM.
- Hardware Hiding: The design must abstract the pins and interface away from the logic such that manipulating the pin mapping or programmable parameters does not change functionality.

- Modes: The system must support the pacemaker bradycardia therapy modes: AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, VVIR.

3.2 Mode Specific Requirements

The pacemaker shall operate according to the following bradycardia pacing modes defined by the pacing, sensing, response, and rate modulation behaviors:

3.2.1 AOO

- The pacemaker should only deliver paces to the atrium.
- The pacemaker should not sense intrinsic atrial or ventricular activity.
- The pacemaker should deliver the paces asynchronously at the programmed rate.
- The pacing rate should remain fixed.

3.2.2 VOO

- The pacemaker should only deliver paces to the ventricle.
- The pacemaker should not sense intrinsic atrial or ventricular activity.
- The pacemaker should deliver the paces asynchronously at the programmed rate.
- The pacing rate should remain fixed.

3.2.3 AAI

- The pacemaker should only deliver paces to the atrium.
- The pacemaker should sense intrinsic atrial activity.
- The pacemaker should inhibit pacing when the detected intrinsic atrial event is above the lower rate limit.
- The pacing rate should remain fixed.

3.2.4 VVI

- The pacemaker should only deliver paces to the ventricle.
- The pacemaker should sense intrinsic ventricular activity.
- The pacemaker should inhibit pacing when the detected intrinsic ventricular event is above the lower rate limit.
- The pacing rate should remain fixed.

3.2.5 AOOR

- The pacemaker should only deliver paces to the atrium.
- The pacemaker should not sense intrinsic atrial or ventricular activity.
- The pacemaker should deliver the paces asynchronously at the programmed rate.
- The pacing rate should be adapted based on inputs from the on-board accelerometer.

3.2.6 VOOR

- The pacemaker should only deliver paces to the ventricle.
- The pacemaker should not sense intrinsic atrial or ventricular activity.
- The pacemaker should deliver the paces asynchronously at the programmed rate.
- The pacing rate should be adapted based on inputs from the on-board accelerometer.

3.2.7 AAIR

- The pacemaker should only deliver paces to the atrium.
- The pacemaker should sense intrinsic atrial activity.
- The pacemaker should inhibit pacing when the detected intrinsic atrial event is above the lower rate limit.
- The pacing rate should be adapted based on inputs from the on-board accelerometer.

3.2.8 VVIR

- The pacemaker should only deliver paces to the ventricle.
- The pacemaker should sense intrinsic ventricular activity.
- The pacemaker should inhibit pacing when the detected intrinsic ventricular event is above the lower rate limit.
- The pacing rate should be adapted based on inputs from the on-board accelerometer.

3.3 Design

3.3.1 System Architecture

The pacemaker Simulink model is implemented using a strictly modular architecture consisting of seven main subsystem references:

- Programmable Parameters Subsystem

- Sensing Subsystem
- Pacemaker Stateflow Logic Subsystem
- Hardware Hiding Subsystem
- Rate Adaptive Subsystem
- Serial Receive Subsystem
- Serial Transmit Subsystem

Each of these is implemented as a Simulink Subsystem Reference rather than a regular subsystem. This design choice ensures that each module exists as an independently editable file. As a result, different team members can safely modify individual subsystems in parallel without modifying the top-level model. This is critical because Simulink models are stored as binary files, which do not merge cleanly using standard version control systems.

By using subsystem references, the top-level Simulink model serves as a modular integration layer rather than the location of core behavior. This facilitates clean version control workflows, allows rapid subsystem swapping or iteration, and enforces separation of concerns and hardware abstraction.

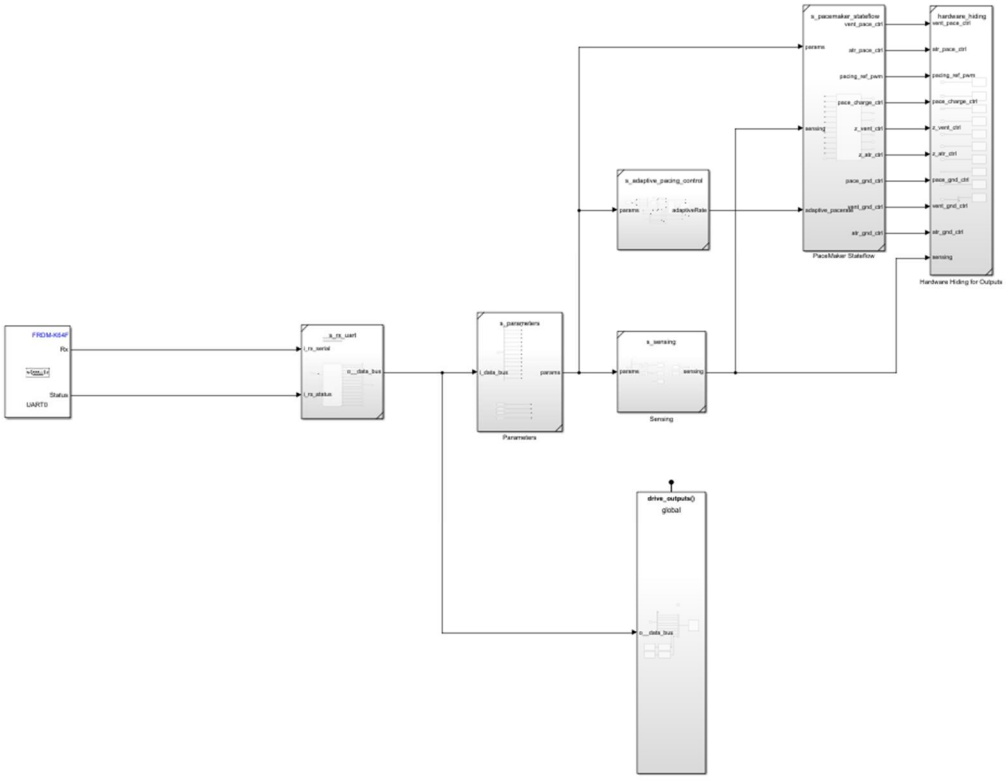


Figure 1: Simulink System Architecture

3.3.2 Programmable Parameters

The pacemaker consists of programmable parameters that define its pacing behavior and timing characteristics described in *Table 1* below:

Table 1: Parameters Used for Each Bradycardia Therapy Mode

Parameter	Nominal	AOO	AAI	VOO	VVI	AOOR	AAIR	VOOR	VVIR
Lower Rate Limit	60 ppm	X	X	X	X	X	X	X	X
Upper Rate Limit	120 ppm	X	X	X	X	X	X	X	X
Maximum Sensor Rate	120 ppm					X	X	X	X
Atrial Amplitude	3.5 V	X	X			X	X		
Ventricular Amplitude	3.5 V			X	X			X	X
Atrial Pulse Width	0.4 ms	X	X			X	X		
Ventricular Pulse Width	0.4 ms			X	X			X	X
Atrial Sensitivity	0.75 mV		X				X		
Ventricular Sensitivity	2.5 mV				X				X
Ventricular Refractory Period	320 ms				X				X
Atrial Refractory Period	250 ms		X				X		
PVARP	250 ms		X				X		
Hysteresis	Off		X		X		X		X
Rate Smoothing	Off		X		X		X		X
Activity Threshold	Med					X	X	X	X
Reaction Time	30 sec					X	X	X	X
Response Factor	8					X	X	X	X
Recovery Time	5 min					X	X	X	X

The programmable parameters subsystem contains all configuration values required by the pacemaker logic.

Table 2: Mapping between Technical and Corresponding Pin Names specific to K64F Microcontroller

Pin	Associated name	Description
Hardware Inputs (from model to board)		
D6	ATR_CMP_REF_PWM	Atrial sensing comparator reference voltage. The atrial sensitivity parameter is divided by 3.3 V and multiplied by 100 to generate a 0–100 PWM duty cycle corresponding to 0–3.3 V.
D3	VENT_CMP_REF_PWM	Ventricular sensing comparator reference voltage (processed identically to atrial sensitivity).
D13	FRONTEND_CTRL	Digital enable signal that activates the sensing front-end circuitry.
Hardware Outputs (signals sensed from board into model)		
D0	ATR_CMP_DETECT	Digital input indicating atrial event detection.
D1	VENT_CMP_DETECT	Digital input indicating ventricular event detection.
Pacing Control Outputs (from model to hardware)		
D8	ATR_PACE_CTRL	Atrial pacing activation switch.
D9	VENT_PACE_CTRL	Ventricular pacing activation switch.
D5	PACING_REF_PWM	PWM reference for pacing pulse amplitude.
D2	PACE_CHARGE_CTRL	Enables charging of the pacing capacitor prior to pulse delivery.
D10	PACE_GND_CTRL	Enables pacing ground return for charge balancing.
D11	ATR_GND_CTRL	Atrial specific ground discharge control.
D12	VENT_GND_CTRL	Ventricular specific ground discharge control.
D4	Z_ATR_CTRL	Atrial impedance control switch (charge balancing / output isolation).
D7	Z_VENT_CTRL	Ventricular impedance control switch.

All these signals are abstracted within the Hardware Hiding subsystem, ensuring the Stateflow pacing logic remains fully independent of physical pin mapping. Likewise, digital sensing signals are received via dedicated comparator detector inputs and abstracted for use by the Stateflow.

3.3.3 Finite State Machine Design

The top-level Stateflow chart selects the active pacing mode based on the mode signal:

Table 3: Pacemaker Modes and their Respective Configuration Value

Mode Configuration	Mode
1	AOO
2	AAI
3	VOO
4	VVI

5	AOOR
6	VOOR
7	AAIR
8	VVIR

Only the corresponding mode's internal states execute, enforcing mode isolation and modularity.

AOO Mode (Asynchronous Atrial Pacing)

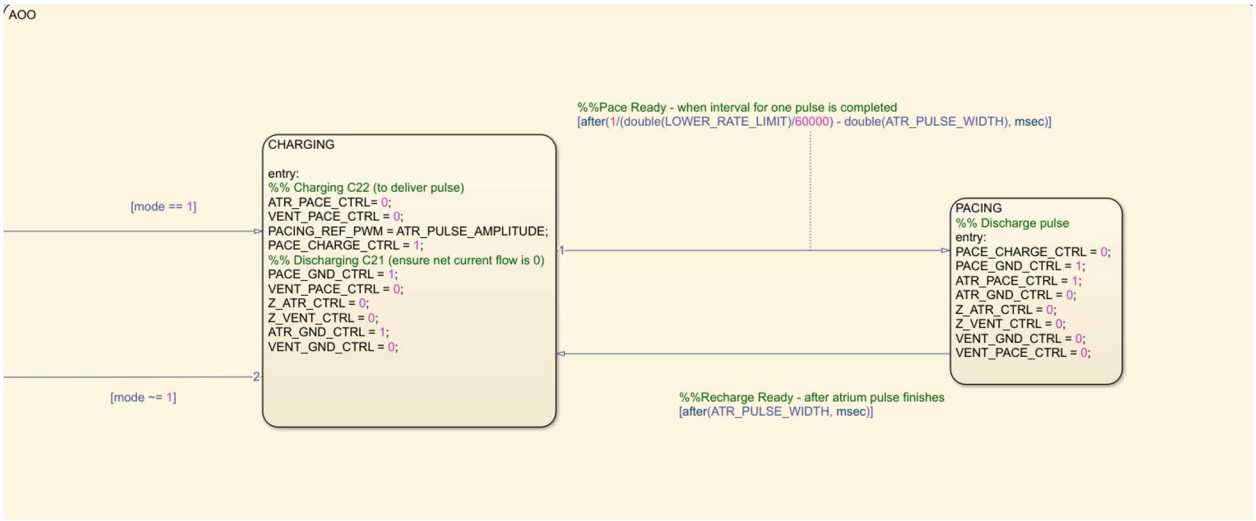


Figure 2: AOO Simulink State

Table 4: State Transition Table for AOO

State	Purpose	Entry Actions	Transition
CHARGING	Charge atrial capacitor and discharge blocking capacitor	ATR_PACE_CTRL=0 PACE_CHARGE_CTRL=1 PACE_GND_CTRL=1 ATR_GND_CTRL=1 PACING_REF_PWM=ATR_PULSE_AMPLITUDE	After 60,000/LRL - ATR_PULSE_WIDTH ms → PACING
PACING	Deliver atrial pulse	ATR_PACE_CTRL=1 PACE_CHARGE_CTRL=0 PACE_GND_CTRL=1 ATR_GND_CTRL=0	After ATR_PULSE_WIDTH ms → CHARGING

VOO Mode (Asynchronous Ventricular Pacing)

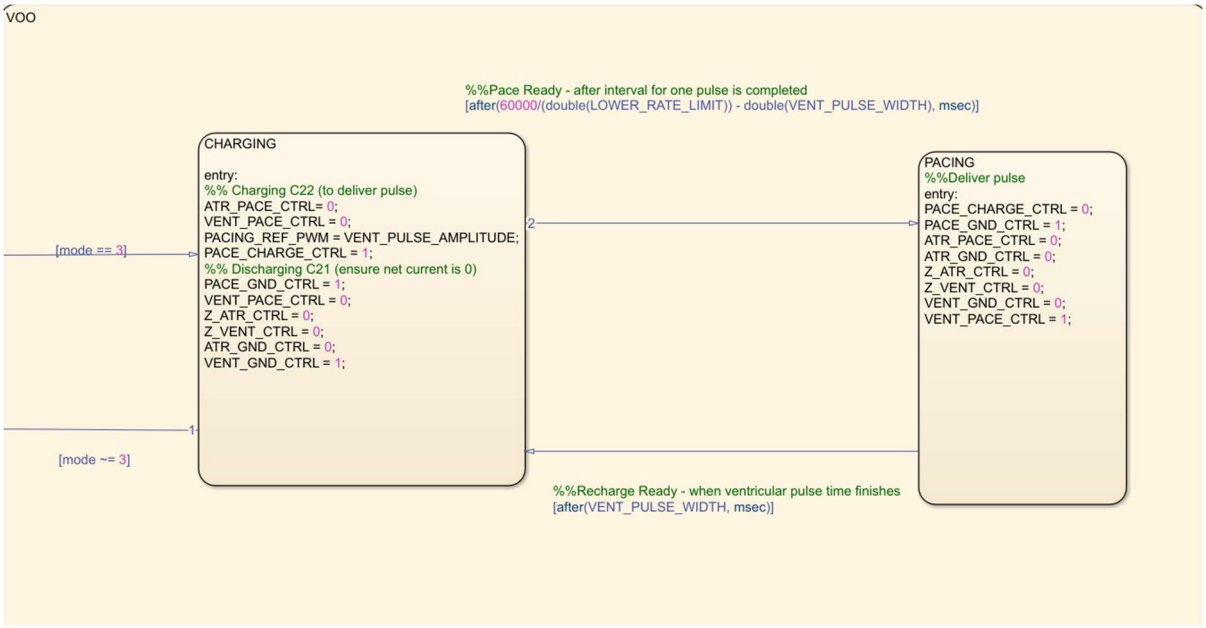


Figure 3: VOO Simulink State

Mirrors the logic for AOO, but ventricular pins are activated (VENT_PACE_CTRL, VENT_GND_CTRL) instead of atrial pins.

States: CHARGING → PACING, with timing determined by Ventricular Pulse Width and LRL.

AAI Mode (Atrial Inhibited Pacing)

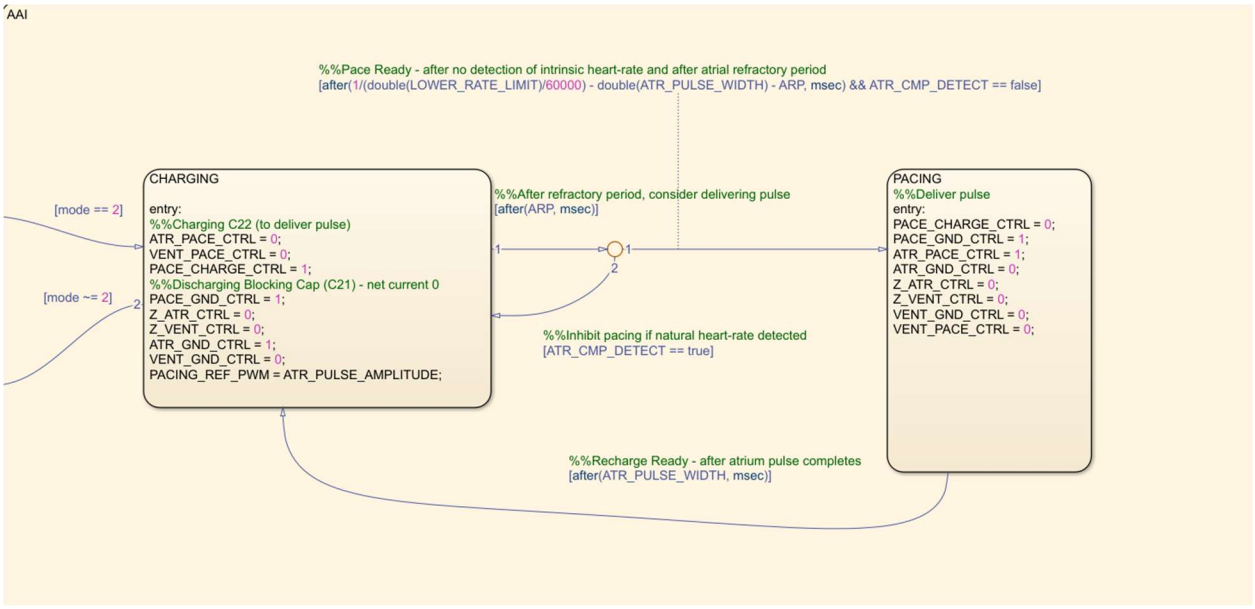


Figure 4: AAI Simulink State

Table 5: State Transition Table for AAI

State	Purpose	Entry Actions	Transition
CHARGING	Charge atrial capacitor, discharge blocking capacitor, monitor intrinsic atrial events	ATR_PACE_CTRL=0 PACE_CHARGE_CTRL=1 PACE_GND_CTRL=1 ATR_GND_CTRL=1, PACING_REF_PWM=ATR_PULSE_AMPLITUDE	After 60,000/LRL - ATR_PULSE_WIDTH - ARP ms **and** ATR_CMP_DETECT==false → PACING; if ATR_CMP_DETECT==true → return to CHARGING
PACING	Deliver atrial pulse	ATR_PACE_CTRL=1 PACE_CHARGE_CTRL=0 PACE_GND_CTRL=1 ATR_GND_CTRL=0	After ATR_PULSE_WIDTH ms → CHARGING

VVI Mode (Ventricular Inhibited Pacing)

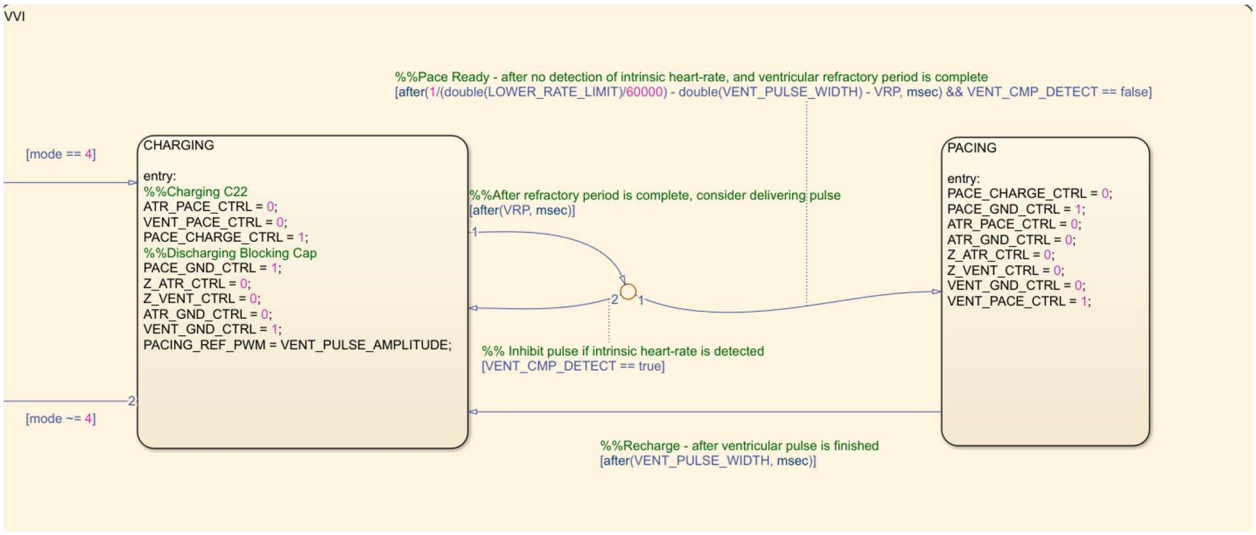


Figure 5: VVI Simulink State

Mirrors AAI logic but monitors ventricular events (VENT_CMP_DETECT) and uses VRP.

States: CHARGING → PACING; CHARGING will inhibit pacing if an intrinsic ventricular event occurs.

AOOR Mode (Asynchronous Rate Adaptive Atrial Pacing)

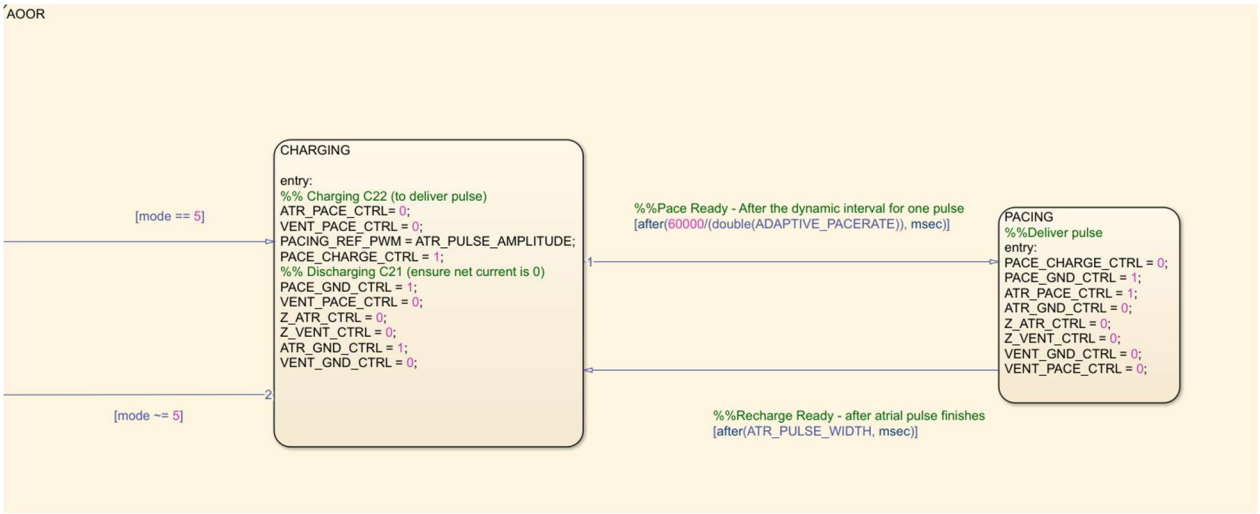


Figure 6: AOOR Simulink State

Table 6: State Transition Table for AOOR

State	Purpose	Entry Actions	Transition
CHARGING	Charge atrial capacitor and discharge blocking capacitor	ATR_PACE_CTRL=0 PACE_CHARGE_CTRL=1 PACE_GND_CTRL=1 ATR_GND_CTRL=1 PACING_REF_PWM=ATR_PULSE_AMPLITUDE	After 60,000/ ADAPTIVE_PACERATE ms → PACING
PACING	Deliver atrial pulse	ATR_PACE_CTRL=1 PACE_CHARGE_CTRL=0 PACE_GND_CTRL=1 ATR_GND_CTRL=0	After ATR_PULSE_WIDTH ms → CHARGING

VOOR Mode (Asynchronous Rate Adaptive Ventricular Pacing)

Figure 7: VOOR Simulink State

Mirrors the logic for AOOR, but ventricular pins are activated (VENT_PACE_CTRL, VENT_GND_CTRL) instead of atrial pins.

States: CHARGING → PACING, with timing determined by Ventricular Pulse Width and ADAPTIVE_PACERATE.

AAIR Mode (Atrial Inhibited Rate Adaptive Pacing)

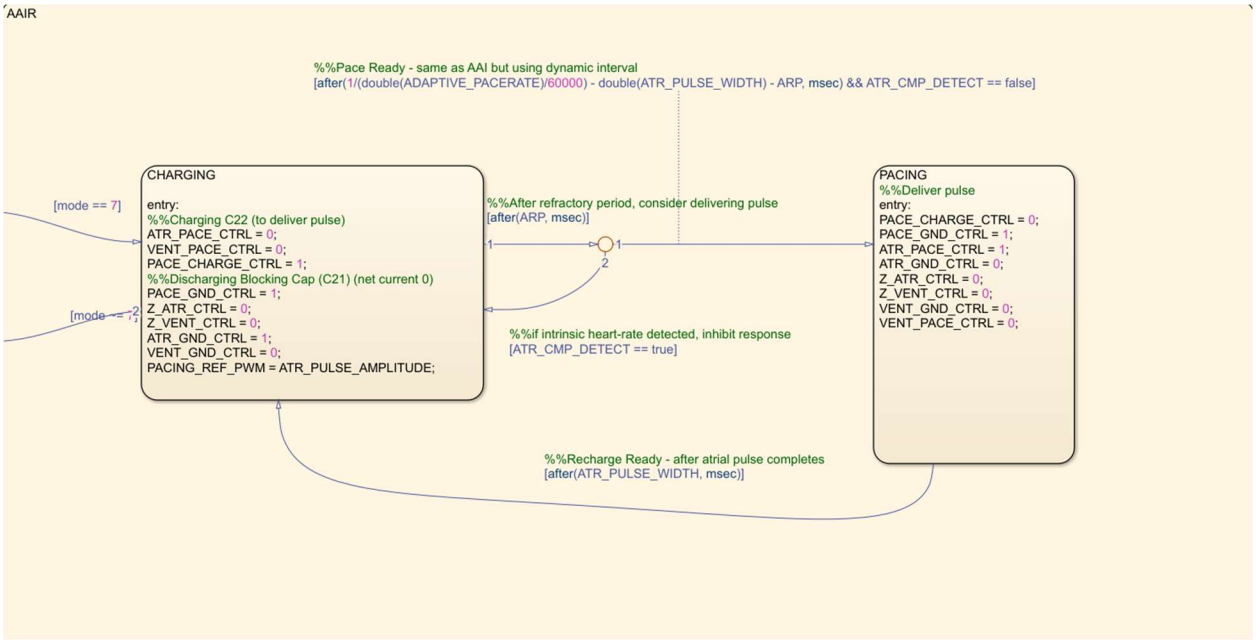


Figure 8: AAIR Simulink State

Table 7: State Transition Table for AAIR

State	Purpose	Entry Actions	Transition
CHARGING	Charge atrial capacitor, discharge blocking capacitor, monitor intrinsic atrial events	ATR_PACE_CTRL=0 PACE_CHARGE_CTRL=1 PACE_GND_CTRL=1 ATR_GND_CTRL=1, PACING_REF_PWM=ATR_PULSE_AMPLITUDE	After 60,000/ ADAPTIVE_PACERATE-ATR_PULSE_WIDTH - ARP ms **and** ATR_CMP_DETECT==false → PACING; if ATR_CMP_DETECT==true → return to CHARGING
PACING	Deliver atrial pulse	ATR_PACE_CTRL=1 PACE_CHARGE_CTRL=0 PACE_GND_CTRL=1 ATR_GND_CTRL=0	After ATR_PULSE_WIDTH ms → CHARGING

VVIR Mode (Ventricular Inhibited Rate Adaptive Pacing)

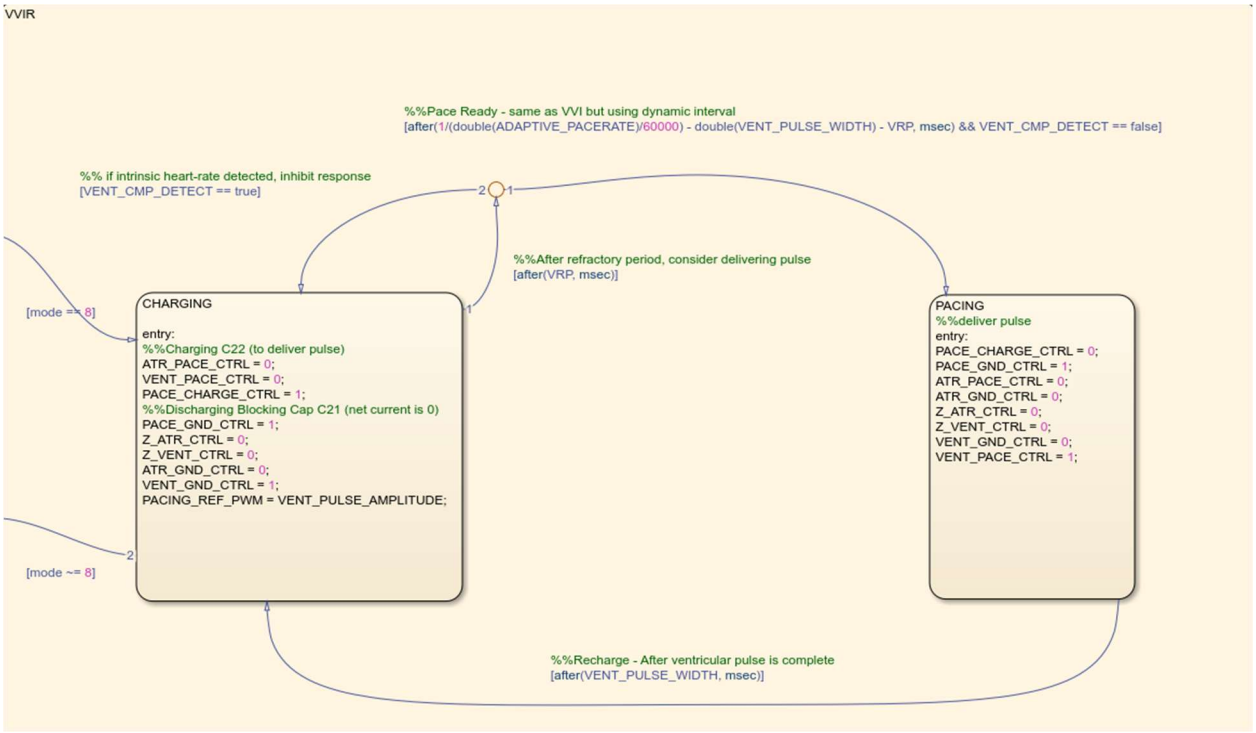


Figure 9: VVIR Simulink State

Mirrors AAIR logic but monitors ventricular events (VENT_CMP_DETECT) and uses VRP, as well as ADAPTIVE_PACERATE.

States: CHARGING → PACING; CHARGING will inhibit pacing if an intrinsic ventricular event occurs.

3.4 Requirements Potential Changes

Potential requirements changes for future iterations include the following:

- Addition of dual-chamber heart-rate therapy modes such as DOOR, DDIR, DDDR, etc.
- Addition of new parameters to accommodate above pacing modes for effective pace-modulation.
- Addition of wireless telemetry and clinician alerting
- Addition of expanded diagnostics and trend reporting

3.5 Design Decision Potential Changes

- Adjust the State Flow model to accommodate the newly added pacing modes for dual chamber pacing.
- Expanding the hardware hiding subsystem for additional output mappings in the future.

- Update the State Flow model to support new dual-chamber pacing modes.
- Expand the hardware subsystem for future output mappings and flexibility.

3.6 Module Description

3.6.1 Sensing Subsystem

The purpose of this module is to provide atrial and ventricular pulse detection from the heart to the pacemaker model, to determine when pacing should occur (or be inhibited) based on respectively set sensitivity thresholds.

Within this module, sensing logic is applied to compare the input signal from the heart to PWM ($V_{ref} = 3.3$ V) reference signals generated for the atrium and ventricular chambers, based on set sensitivity measurements (hardware mapped within this module). Based on the comparator logic, detection is determined via Boolean output from hardware-mapped pins on the microcontroller (ATR_CMP_DETECT and VENT_CMP_DETECT). These variables are passed as input under the input bus group, "sensing", to be used within the Pacemaker Stateflow block. This block also receives input from the Parameters block to reference atrium and ventricular sensitivity thresholds.

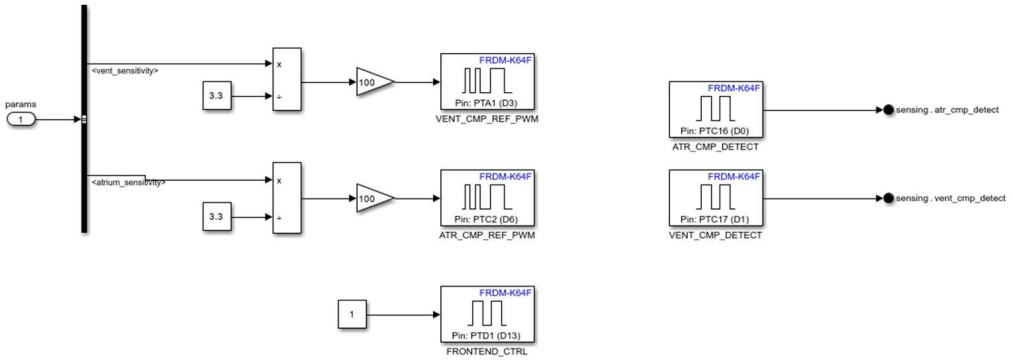


Figure 10: Sensing Subsystem

3.6.2 Parameters Subsystem

The purpose of this module is to contain the programmable parameters for pace-making functionality. These parameters are set via Serial Communication (UART with DCM) /constant blocks based on nominal values provided in *Table 1*. These inputs are mapped to the bus input group, "params", which is then passed to the Sensing (vent_sensitivity and atrium_sensitivity) and Pacemaker Stateflow blocks.

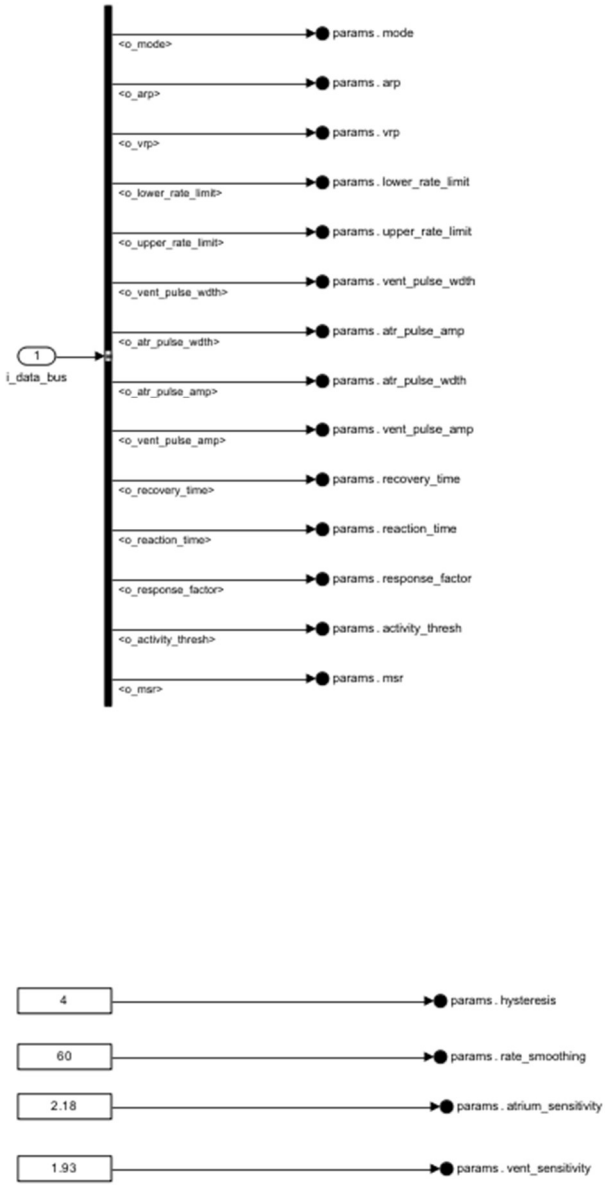


Figure 11: Programmable Parameters Subsystem

3.6.3 Pacemaker Stateflow Subsystem

The purpose of this module is to contain the software logic in implementing the bradycardia therapy modes that the pacemaker is responsible for. Within this module, each pacing mode is implemented as an FSM to deliver pacing when needed through time-based transition logic. These states utilize input variables from “params” as well as “sensing”, to safely deliver pacing when required. Additionally, output variables from this module are passed as input to the Hardware Hiding for Outputs block, delivering signals via control lines in the hardware shield while keeping the core logic hardware-free.

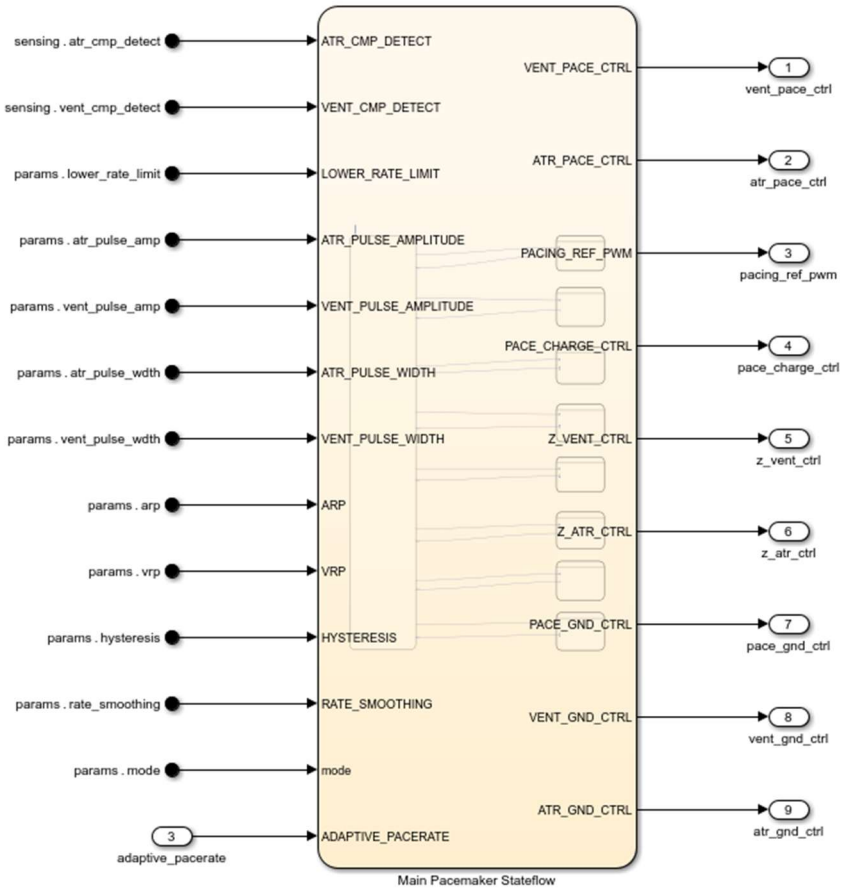


Figure 12: Pacemaker Stateflow Subsystem

3.6.4 Hardware Hiding for Outputs Subsystem

The purpose of this module is to form an abstract layer between the software logic and the hardware-mapped outputs. Within this module, the outputs passed from the Stateflow block are mapped to their corresponding output pin on the microcontroller, as described from the Hardware inputs/outputs table (pacing control outputs). Within this module, conversion from software to hardware-specifics takes place, allowing for portability and safety-critical testing to be performed without altering key aspects of the model.

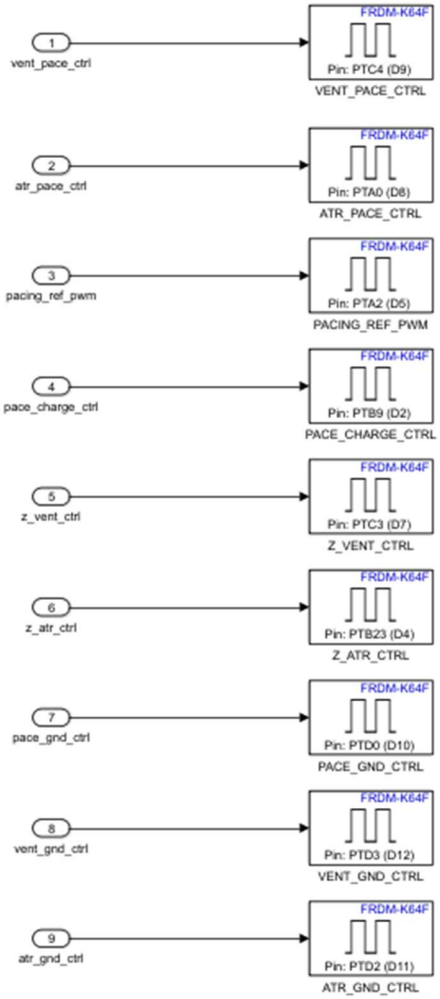


Figure 13: Hardware Hiding Subsystem

3.6.5 Adaptive Pacing Control Subsystem

The purpose of this module is to provide AOO, VOO, AAI, and VVI functionality based on a dynamically adjusted rate (hence R in the national letter code). This is accomplished using the FRDM-K64F on-board accelerometer, which measures acceleration as a 3-dimensional vector (x, y, z). It is then demultiplexed (split into 3 scalar signals), in which each signal has its magnitude squared and is then summed together. This results in a scalar output representative of each acceleration component, ensuring accelerometer motion encapsulates all degrees of freedom. This data is then used to compute a moving-average with a discrete-sample window (MATLAB function). The moving average works as an FIR low-pass filter, effectively smoothing out the signal formed from each sample point, to eliminate noise commonly associated with accelerometer readings. The moving-average computed for each data sample is then mapped to an activity level, by applying scaling and rounding, resulting in an integer-mapped output. This activity level is then utilized in another function to calculate the desired pacing rate, along with

inputs from *Table 1* such as Lower-Rate Limit, Maximum Sensor Rate, Response Factor, and Activity Threshold. This allows for the desired pacing rate to be determined through checking the current activity level against the threshold (greater), in which a linear-rate response curve is used to obtain a desired setting along the curve, bounded by the Lower-Rate-Limit and Maximum Sensor Rate. Note that the Response Factor governs the slope of the curve, resulting in higher rates for higher response factors. Lastly, due to heart physiology, the desired pace rate can only be achieved through a gradual increase in the current pace-rate. To achieve this, a StateFlow chart is implemented which takes the target pace rate, Lower Rate Limit, and current pacing mode as inputs, in addition to other parameters from *Table 1* such as Reaction Time and Recovery Time. Within the chart, a decision-based FSM is implemented to compare the current rate to the target and appropriately calculate the step-size increase/decrease (ppm per ms) using the Recovery Time and Reaction Time parameters, which set the duration that the step-size is applied for. As a result, the current pace-rate is adjusted dynamically, and is then passed as input to the R-dependent modes to conduct appropriate bradycardia therapy.

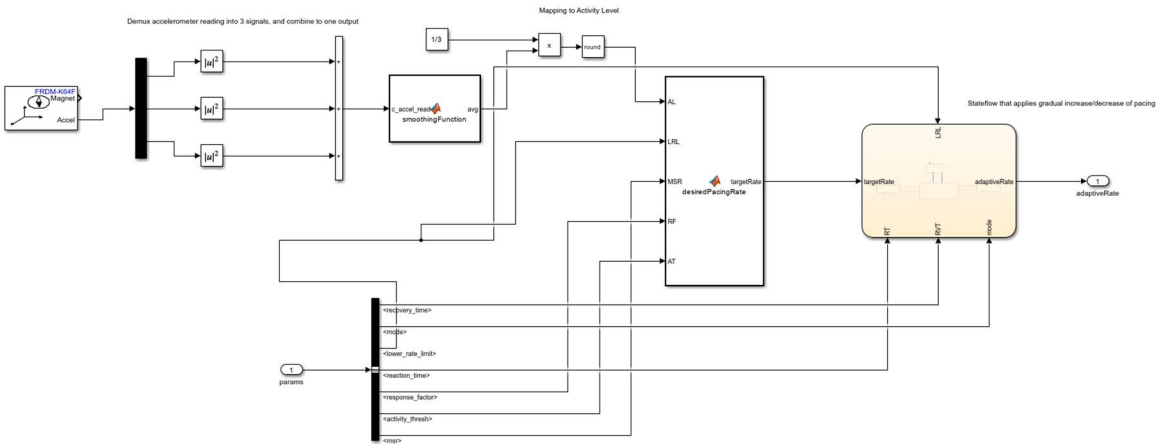


Figure 14: Rate Adaptive Subsystem

s_adaptive_pacing_control ▸ MATLAB Function

```
%Applies a Moving Average Filter (FIR low-pass)

function avg = smoothingFunction(c_accel_read)
    %initialize storage for past samples
    persistent buffer index count;

    N = 20; %window size of 20 samples

    %on startup, fill the buffer and initialize iterator
    if isempty(buffer)
        buffer = zeros(1, N)
        index = 1;
        count = 0;
    end

    buffer(index) = c_accel_read; %add the recent sample to the window
    %update index circularly (shift window rightwards)
    index = mod(index, N) + 1;

    count = min(count + 1, N); %coefficient are adjusted dynamically

    avg = sum(buffer) / count; %returns the average (smoothed output)

end
```

Figure 15: Rate Adaptive Pacing Subsystem - Smoothing Function

s_adaptive_pacing_control ▸ MATLAB Function1

```
%Maps desired pacing rate between LRL and MSR

function targetRate = desiredPacingRate(AL, LRL, MSR, RF, AT)
    % no signficant change in activity level, clamp to LRL
    if AL < AT
        targetRate = LRL;
    else
        % apply a linear rate-transformation curve (Tutorial-4)
        targetRate = min(LRL + (AL - AT) * RF, MSR);
    end
end

```

Figure 16: Rate Adaptive Pacing Subsystem - Linear Rate Mapping

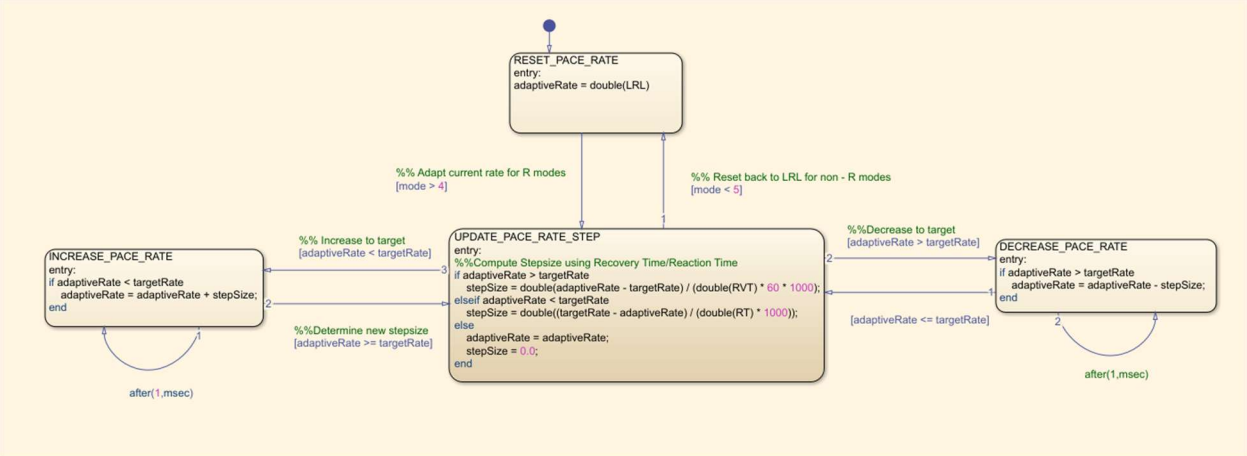


Figure 17: Rate Adaptive Pacing Subsystem – Time Rate Transformation FSM for Gradual Increase/Decrease

3.6.6 Serial Receive Subsystem

This Stateflow-based serial communication subsystem functions as the protocol engine between the DCM and the pacemaker model. It continuously monitors the UART receive interface, waits for a complete incoming frame, and then verifies the synchronization byte before interpreting the instruction code. When a SET_PARAMS command is detected, the subsystem reads each byte of the received frame and updates the corresponding pacing parameters by writing them directly to the model outputs. These parameters include the pacing mode, rate limits, pulse amplitudes and widths, refractory periods, and rate-responsive settings. If a transmit instruction is received, the subsystem activates the transmit branch and calls the UART transmit block to send data back to the DCM. During initialization it loads safe default parameter values, and during operation it repeatedly processes incoming frames in a simple and deterministic loop that keeps the pacemaker model synchronized with the DCM.

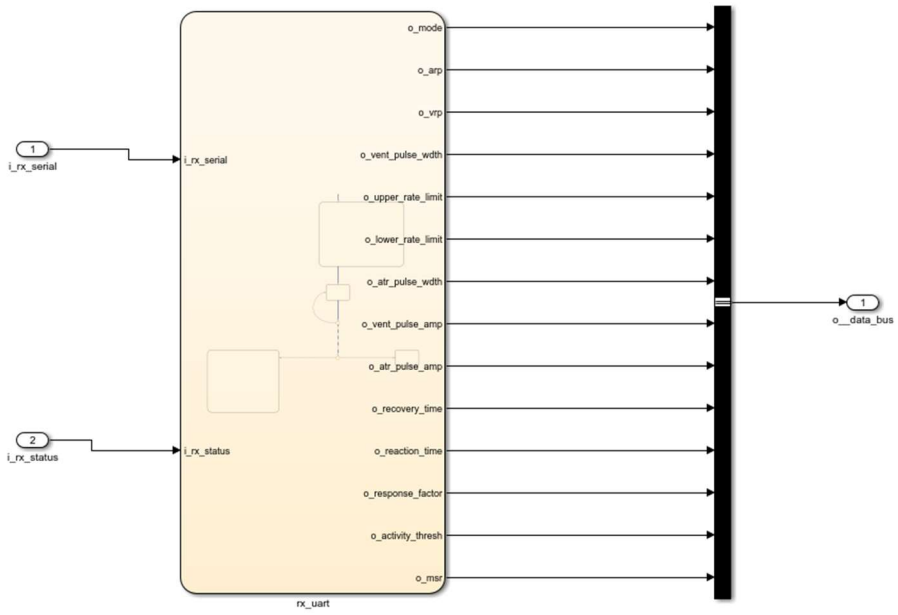


Figure 18: Serial Receive Subsystem

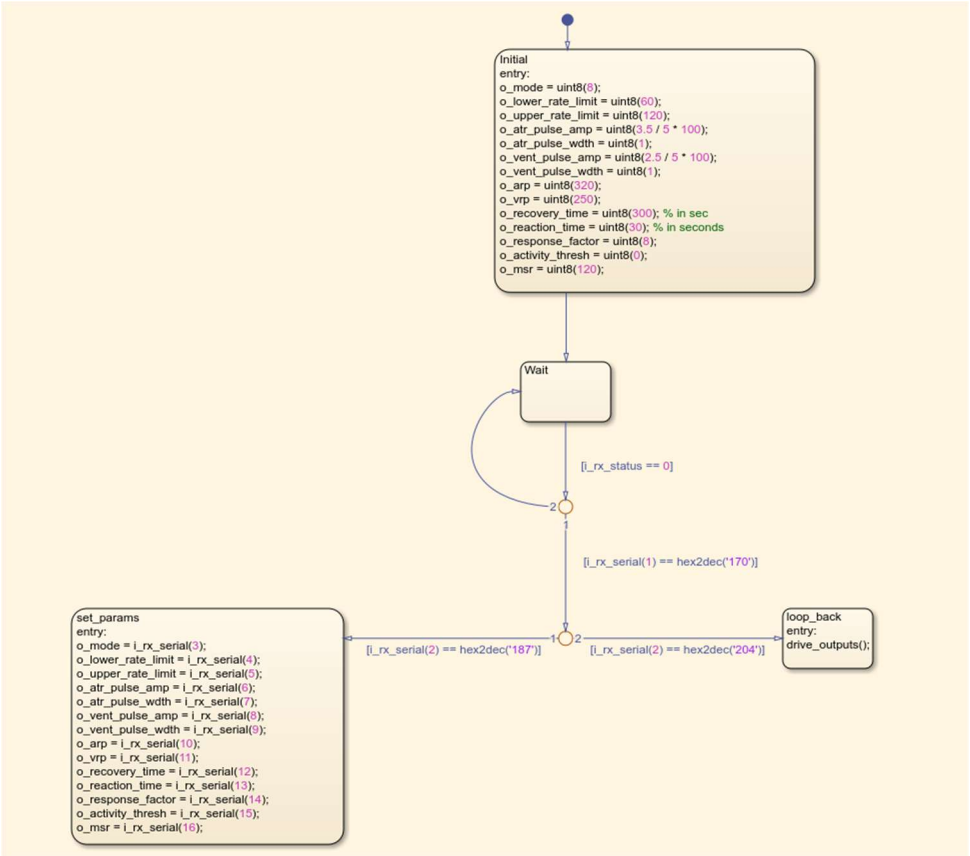


Figure 19: Initialize and Set Parameters FSM

3.6.7 Serial Transmit Subsystem

This transmit subsystem is a simple packaging unit that collects selected pacemaker parameters and analog signals, combines them into a single vector through a multiplex block, and forwards that vector to the Serial Send block on the UART port. It is invoked by the Stateflow chart whenever the received instruction byte is 0xCC, allowing the pacemaker model to send its current configuration or telemetry data back to the DCM in a compact and consistent format.

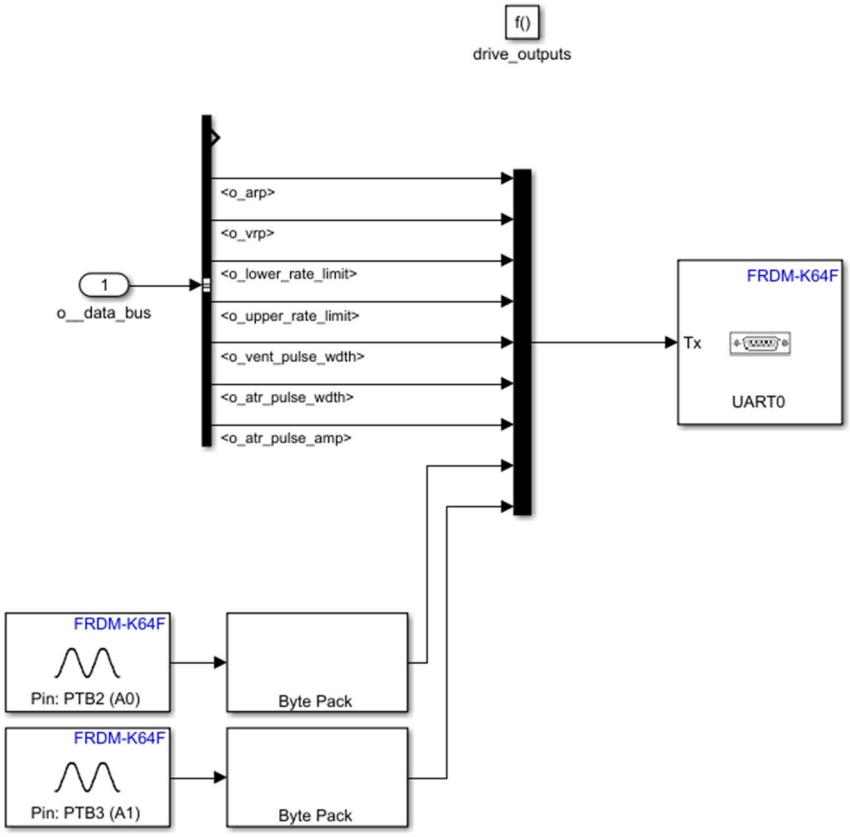


Figure 20: Serial Transmit Subsystem

3.7 Testing

Testing was conducted using the provided pacemaker board and the HeartView desktop application to observe pacing signals. The following tests were performed:

- Mode Switching: Verified correct pacing behavior for each mode by observing the pacing signal output.
- Heart Rate Fluctuation: Verified correct sensing/pacing behavior for changes in BPM (Under and over the lower rate limit).

- Activity Fluctuation: Confirmed correct sensing/pacing behavior for changes in activity detected by the pacemaker.
- Parameter Adjustments: Tested the system's response to changes in pulse width, amplitude, and rate limits.
- Pin Mapping Validation: Confirmed that pin mapping abstraction worked correctly, with the correct hardware interface mapped without modifying the core model.

Table 8: AOO Test 1

Pacemaker Parameter	Input Conditions
Mode	AOO
Atrial Pulse Width	1 ms
Atrial Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Heartview Parameter	Input Conditions
Natural Atrium	On
Natural Ventricle	Off
Natural Heart Rate	60 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker periodically paces the atrium while the simulated atrium beats.

Actual Output: The pacemaker periodically paces the atrium while the simulated atrium beats.

Result: Pass

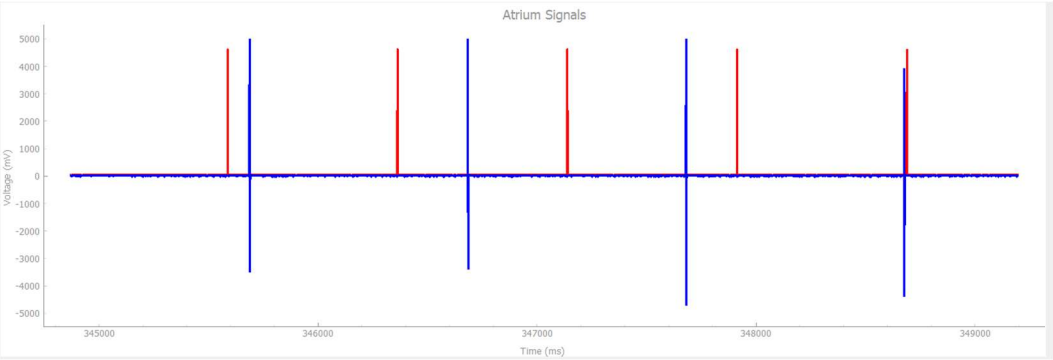


Figure 21: Heartview Output for AOO Test

Table 9: VOO Test 1

Pacemaker Parameter	Input Conditions
Mode	VOO
Ventricular Pulse Width	1 ms

Ventricular Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Heartview Parameter	Input Conditions
Natural Atrium	Off
Natural Ventricle	On
Natural Heart Rate	60 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker periodically paces the ventricle while the simulated ventricle beats.

Actual Output: The pacemaker periodically paces the ventricle while the simulated ventricle beats.

Result: Pass

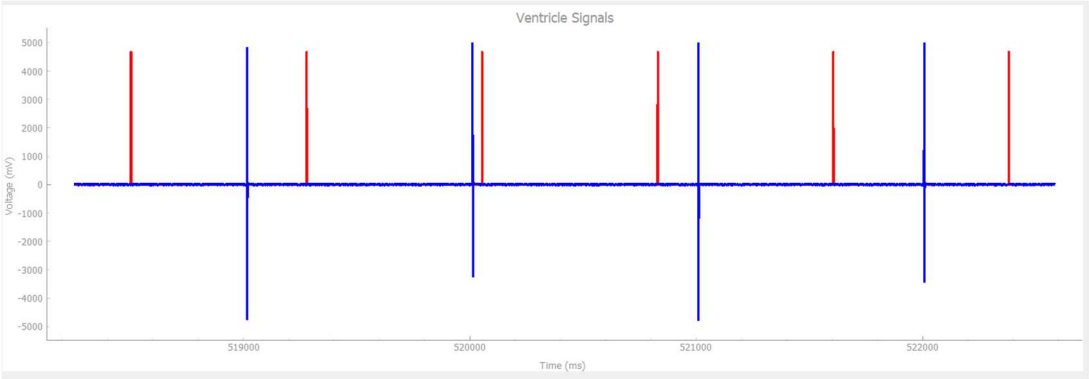


Figure 22: Heartview Output for VOO Test 1

Table 10: AAI Test 1

Pacemaker Parameter	**Input Conditions**
Mode	AAI
Atrial Pulse Width	1 ms
Atrial Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Atrial Refractory Period	250 ms
Heartview Parameter	Input Conditions
Natural Atrium	On
Natural Ventricle	Off
Natural Heart Rate	30 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker detects that the simulated atrium is beating below 60 bpm and paces the atrium.

Actual Output: The pacemaker detects that the simulated atrium is beating below 60 bpm and paces the atrium.

Result: Pass

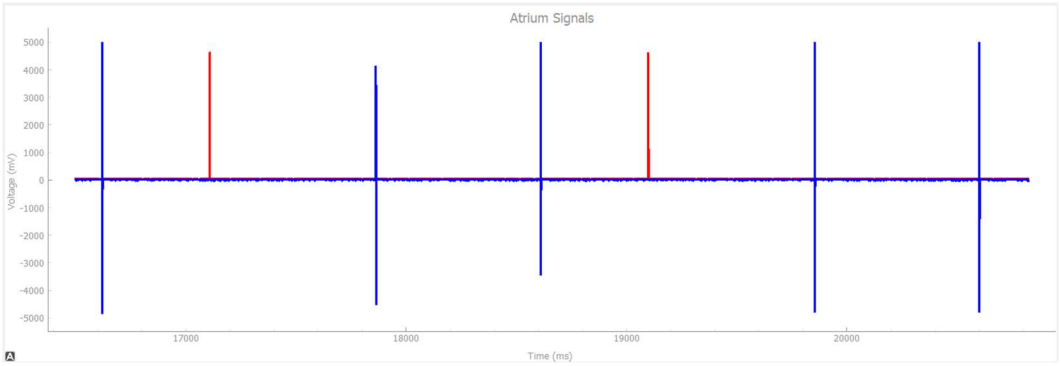


Figure 23: Heartview Output for AAI Test 1

Table 11: AAI Test 2

Pacemaker Parameter	**Input Conditions**
Mode	AAI
Atrial Pulse Width	1 ms
Atrial Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Atrial Refractory Period	250 ms
Heartview Parameter	Input Conditions
Natural Atrium	On
Natural Ventricle	Off
Natural Heart Rate	180 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker detects that the simulated atrium is beating above 60 bpm and does not pace the atrium.

Actual Output: The pacemaker detects that the simulated atrium is beating above 60 bpm and does not pace the atrium.

Result: Pass

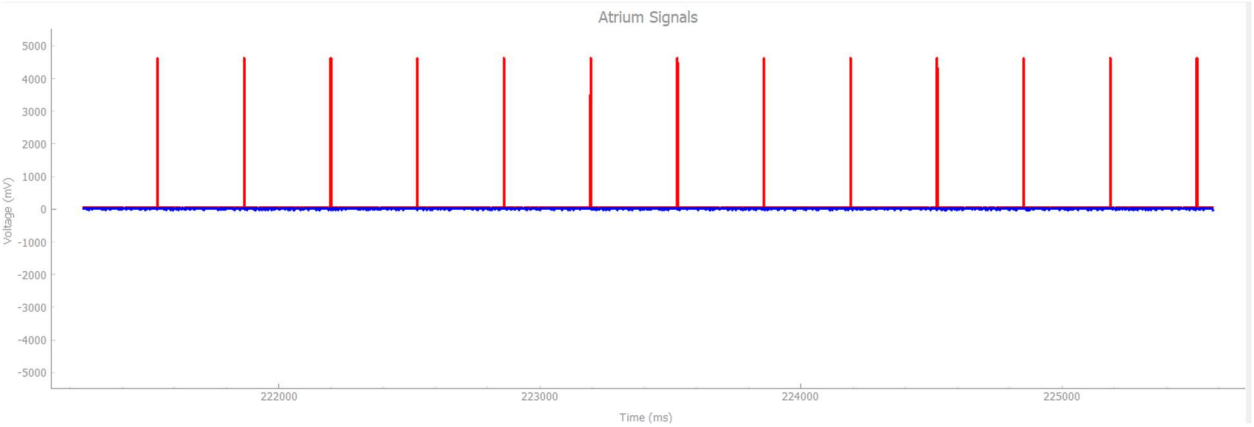


Figure 24: Heartview Output for AAI Test 2

Table 12: VVI Test 1

Pacemaker Parameter	**Input Conditions**
Mode	VVI
Ventricular Pulse Width	1 ms
Ventricular Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Ventricular Refractory Period	320 ms
Heartview Parameter	Input Conditions
Natural Atrium	Off
Natural Ventricle	On
Natural Heart Rate	30 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker detects that the simulated ventricle is beating below 60 bpm and paces the ventricle.

Actual Output: The pacemaker detects that the simulated ventricle is beating below 60 bpm and paces the ventricle.

Result: Pass

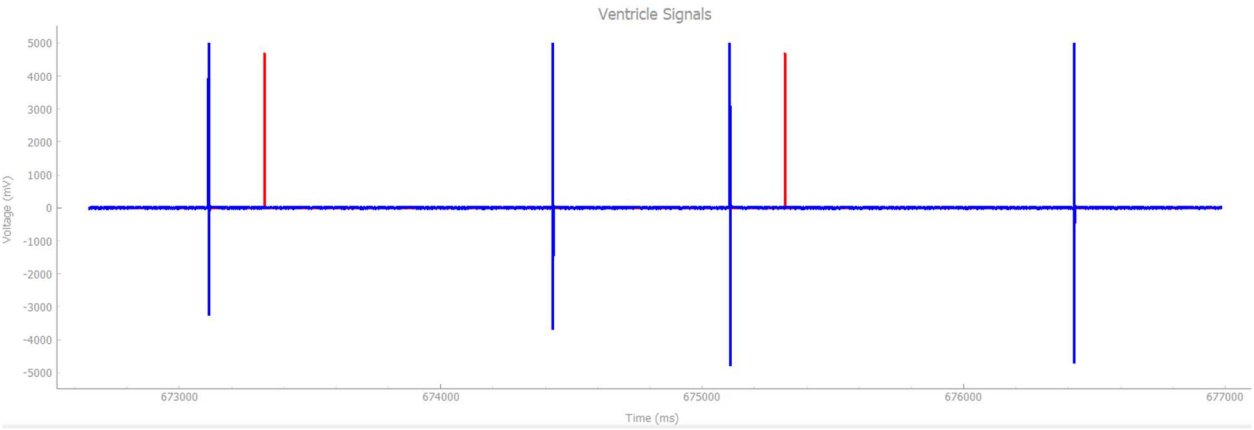


Figure 25: Heartview Output for VVI Test 1

Table 13: VVI Test 2

Pacemaker Parameter	**Input Conditions**
Mode	VVI
Ventricular Pulse Width	1 ms
Ventricular Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Ventricular Refractory Period	320 ms
Heartview Parameter	Input Conditions
Natural Atrium	Off
Natural Ventricle	On
Natural Heart Rate	180 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker detects that the simulated ventricle is beating above 60 bpm and does not pace the atrium.

Actual Output: The pacemaker detects that the simulated ventricle is beating above 60 bpm and does not pace the atrium.

Result: Pass

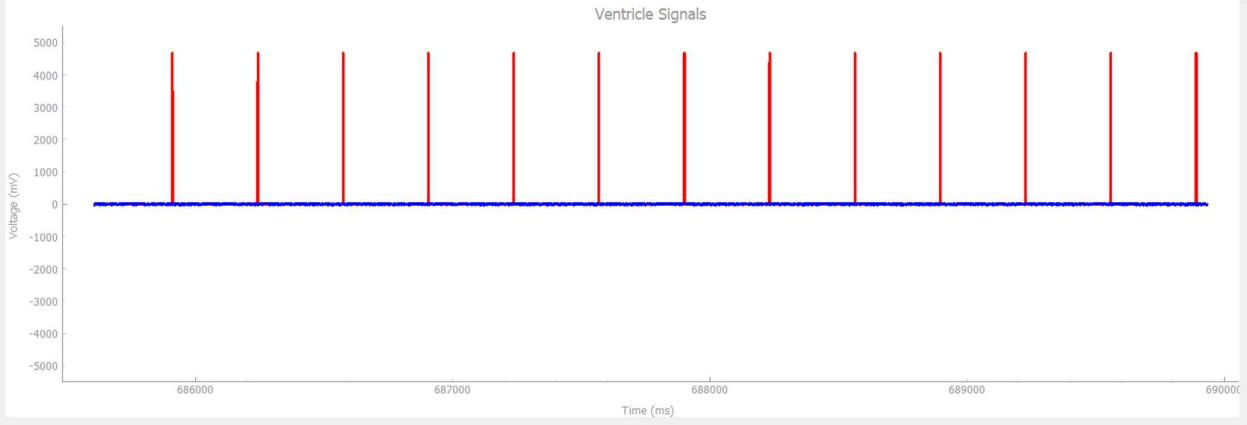


Figure 26: Heartview Output for VVI Test 2

Table 14: AOOR Test

Pacemaker Parameter	**Input Conditions**
Mode	AOOR
Atrial Pulse Width	1 ms
Atrial Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Heartview Parameter	Input Conditions
Natural Atrium	On
Natural Ventricle	Off
Natural Heart Rate	60 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker periodically paces the atrium while the simulated atrium beats then the pacing interval increases as the pacemaker is shaken.

Actual Output: The pacemaker periodically paces the atrium while the simulated atrium beats, but the pacing interval does not increase when the pacemaker is shaken.

Result: Fail

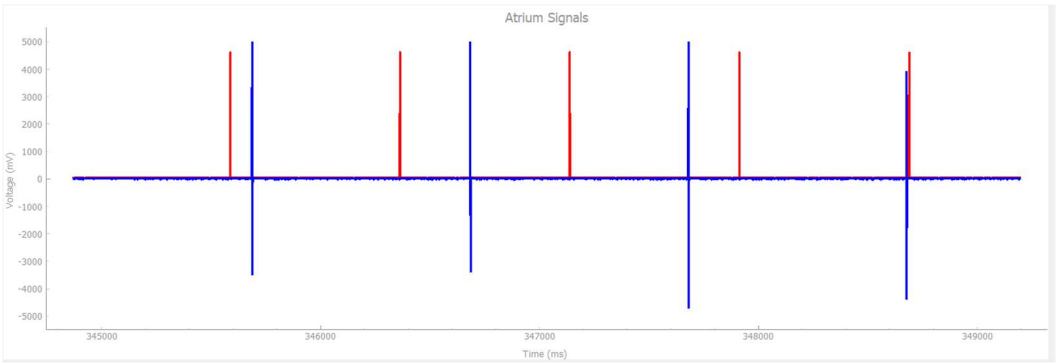


Figure 27: Heartview Output for AOOR Test

Table 15: VOOR Test

Pacemaker Parameter	**Input Conditions**
Mode	VOOR
Ventricular Pulse Width	1 ms
Ventricular Pulse Amplitude	3.5 V
Lower Rate Limit	60 ppm
Heartview Parameter	**Input Conditions**
Natural Atrium	Off
Natural Ventricle	On
Natural Heart Rate	60 bpm
Natural AV Delay	30 ms

Expected Output: The pacemaker periodically paces the ventricle while the simulated ventricle beats then the pacing interval increases as the pacemaker is shaken.

Actual Output: The pacemaker periodically paces the ventricle while the simulated ventricle beats, but the pacing interval does not increase when the pacemaker is shaken.

Result: Fail

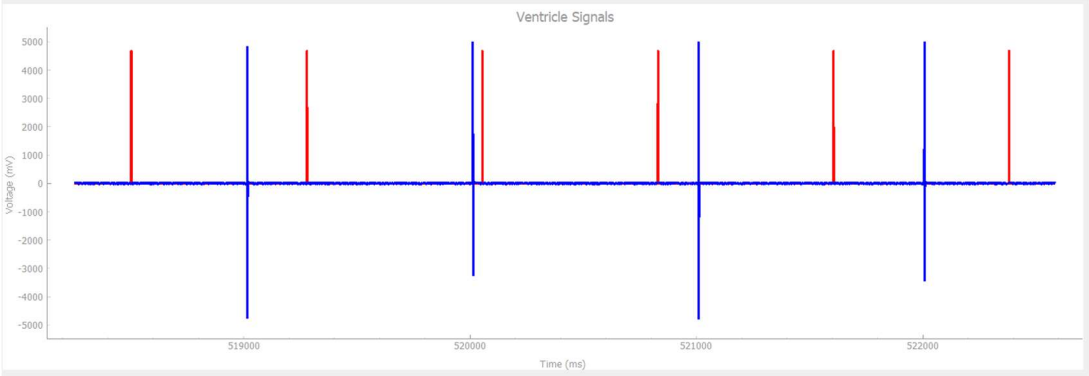


Figure 28: Heartview Output for VOOR Test

4 DCM

4.1 Requirements

The following requirements apply to the DCM component:

- The DCM should provide a login and registration interface allowing up to 10 users.
- The DCM should display a dashboard view after a successful login.
- The DCM should support displaying and selecting one of the following pacing modes: AOO, VOO, AAI, VVI, AOOR, AAIR, VOOR, VVIR.
- The DCM should allow the user to configure all programmable parameters.
- The DCM should display a summary of the currently selected values.
- The DCM should transmit information to and receive from the pacemaker via UART serial communication.
- The DCM should display egram data received from the pacemaker when the user chooses to do so (for either ventricle, atrium, or both).

In addition to the requirements listed in Section 4.1, the Deliverable-2 implementation of the DCM introduces the following functional and non-functional requirements:

- DCM should be implemented as a Python/Tkinter desktop application, rather than a purely web based React front end.
- The DCM shall persist in user accounts and per-user pacemaker settings using local JSON files, rather than browser local Storage.
- The DCM shall support up to 50 registered users, with hashed password storage, while still enforcing a maximum number of accounts.
- The DCM shall remember the last logged-in user and automatically restore their session when the application is reopened.
- DCM should maintain a per-user Pacemaker Settings profile, storing the last programmed values for each pacing mode.

- The DCM shall provide a serial communication layer over UART to enumerate available COM ports, connect/disconnect from a selected port, perform a lightweight connection test, and send compact parameter frames to the pacemaker hardware.
- The DCM shall provide infrastructure for egram streaming, reading raw samples from the serial port in a background thread and pushing them into a queue for display as real-time atrial/ventricular egrams in the UI.

4.2 Design

4.2.1 Software Structure

The DCM follows a modular front-end architecture, where each screen is implemented as an independent component. The core structure is:

- Authentication Component – manages login and registration logic (up to 10 users).
- Dashboard Component – displays high-level device state and navigation links.
- Connection Modal – overlays on the dashboard to simulate pacemaker connection.
- Pacing Modes Component – lists all modes and allows mode activation.
- Parameters Component – provides UI controls for all programmable parameters.
- Shared State – in-memory UI state stores the selected mode, connection status, and current parameter values.

4.2.2 Component Interaction/UI Workflow

The DCM is implemented as a front-end React application and serves as the graphical software layer through which a user configures and interacts with the pacemaker. The interface is divided into four logical modules: (1) authentication, (2) dashboard and device connection, (3) pacing mode selection, and (4) programmable parameter configuration. These modules map directly to the functional requirements listed in Section 2.2 and represent the clinician workflow from login to final parameter adjustment.

The authentication module enforces a maximum of ten registered users and provides access control before any device operations can be performed. Upon login, the dashboard module displays the connection state and current pacemaker mode, and it provides navigation to other screens. The connection modal embedded within the dashboard simulates pacemaker connectivity using selectable COM ports, which will later be bound to real serial communication in Deliverable 2.

The pacing-mode module displays all four Deliverable-1 modes (AOO, VOO, AAI, VVI) together with chamber/sensing/response characteristics, allowing the clinician to activate the desired mode. The parameter configuration module presents eight programmable parameters, each adjustable by sliders, numeric step controls, or direct typed input. These values are summarized in real time to confirm intended configuration before transmission in future deliverables.

As the project progressed beyond the Deliverable-1 prototype, it became clear that the original React-based DCM, while visually polished and modular, introduced significant software complexity that was not aligned with the constraints of the course environment or the required hardware integration. The web implementation made us to be dependant on multiple application layers, which made reliable communication with the pacemaker hardware increasingly difficult to maintain, and especially to debug during the development process.

Based on our experiences during the first phase, particularly with serial communication testing, UI state persistence, and the addition of new rate-responsive parameters, the overall software stack was becoming substantially more complicated. With the available time frame and the need for dependable hardware functionality, the team made a deliberate decision to migrate to a simpler and more robust architecture.

The updated DCM is a Python/Tkinter desktop application designed around clarity, reliability, and maintainability. It provides:

- Direct UART access without browser restrictions or external bridges
- A single-language, single-runtime environment instead of a multi-layer build pipeline
- Simple, predictable persistence using JSON files
- Straightforward debugging tools, such as full UART logs and print-level introspection
- A leaner UI that exposes only essential pacemaker configuration features, reducing the risk of UI-induced errors during hardware testing

This “bare-bones” interface was chosen intentionally: the priority for Deliverable-2 was to ensure that parameter configuration, validation, and hardware communication worked flawlessly rather than focusing on visual aesthetics or advanced user experience features.

4.2.3 Design-to-Requirements

The DCM design directly satisfies the requirements:

- The Authentication Component satisfies the requirement for login and registration, including limiting the system to 10 users.
- The Dashboard and Connection Modal satisfy the requirement to simulate pacemaker connection status and allow the user to select a communication port.
- The Pacing Modes Component satisfies the requirement to display and activate the four Deliverable-1 pacing modes (AOO, VOO, AAI, VVI).
- The Parameters Component satisfies the requirement to configure all programmable parameters through multiple input methods (slider, stepper arrows, and manual entry).
- The front-end-only implementation satisfies the Deliverable-1 constraint that no real hardware connection, persistence, or telemetry is required at this stage.

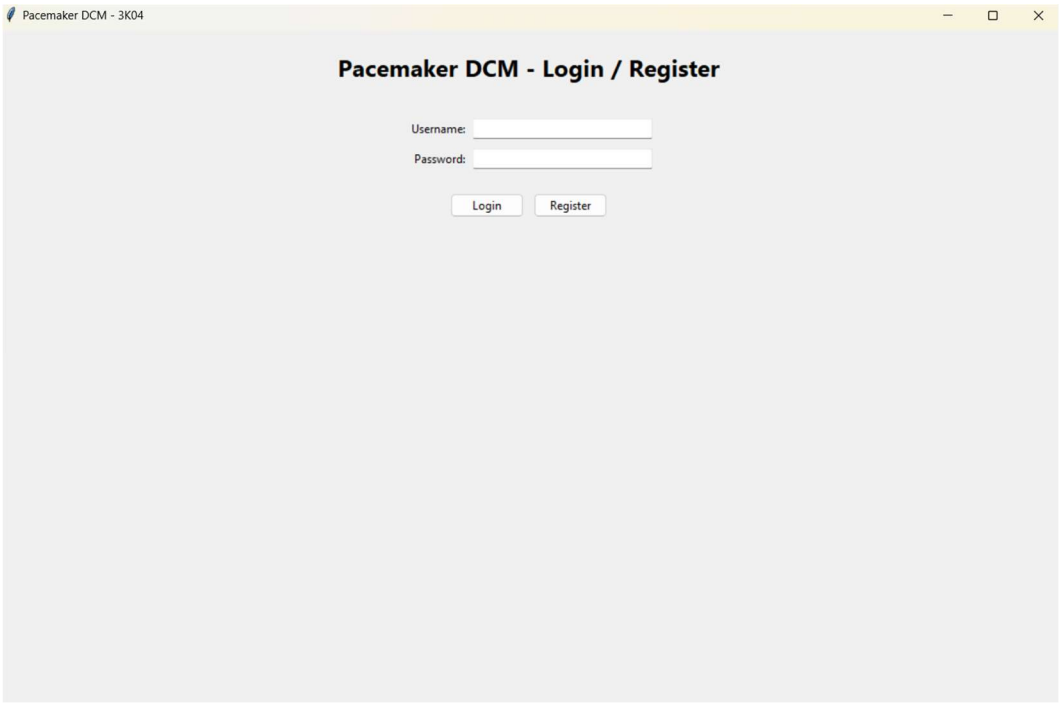


Figure 29: Authentication. The DCM enforces a local maximum of 10 users for D1; a successful login routes to the Dashboard.

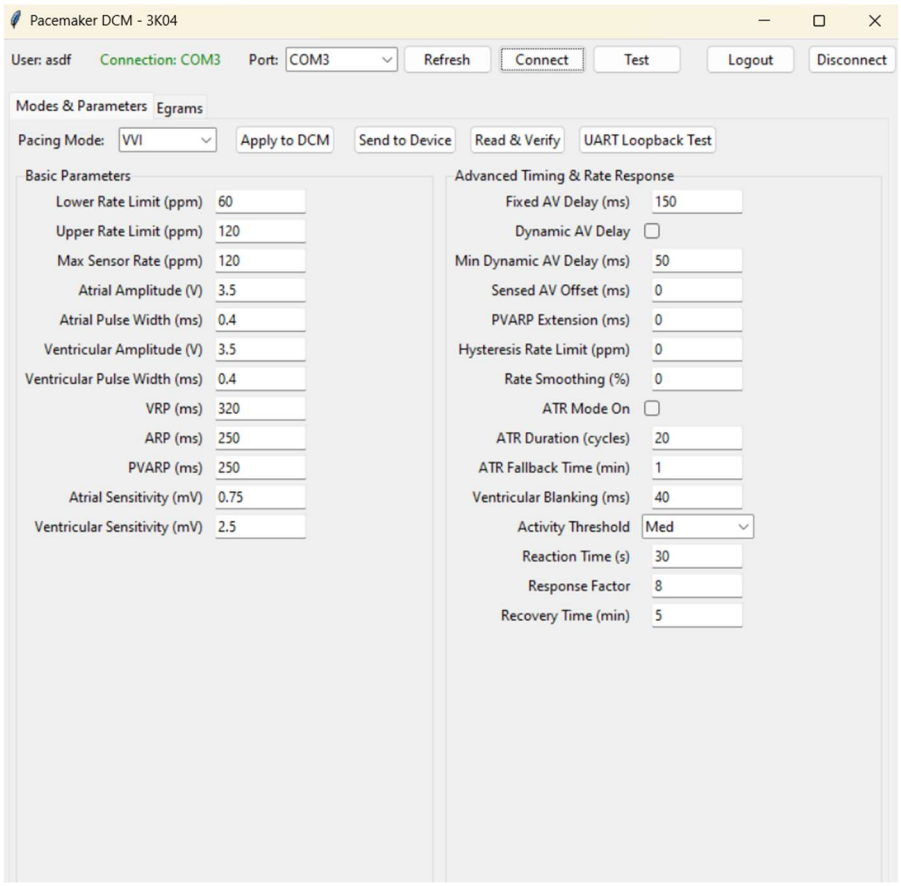


Figure 30: Dashboard (Connected). Connection status, current mode, battery level, and static device info are visible post-login; navigation to Modes/Parameters is available.

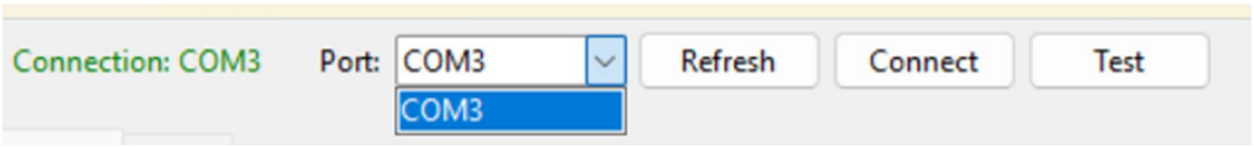


Figure 31: Pacemaker Connection Modal. Users select a COM port and test the (simulated) connection; success toggles the global status to Online.

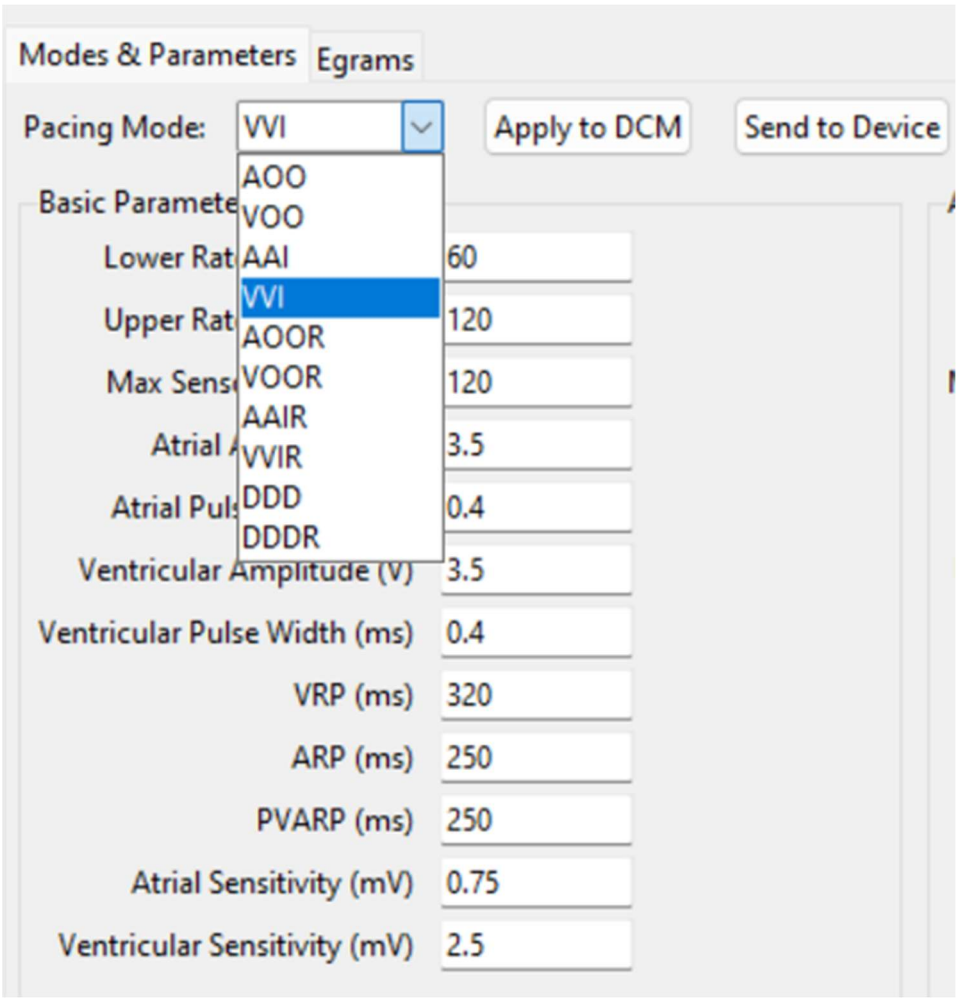


Figure 32: Pacing Modes. All Deliverable 1 modes are presented with chamber/sensing/response; the active mode is highlighted and re-apply is disabled.

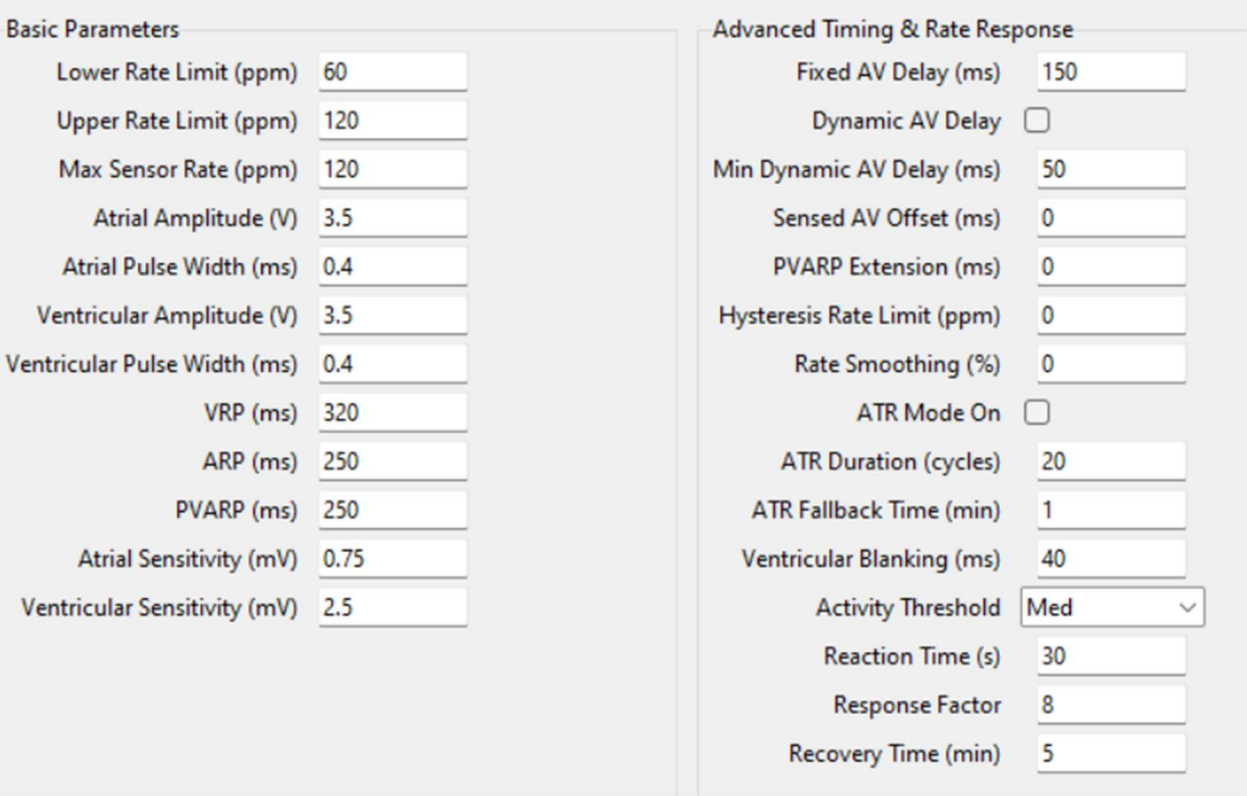


Figure 33: Programmable Parameters. LRL, URL, AA, APW, VA, VPW, VRP, ARP are adjustable via slider, increment/decrement arrows, or direct numeric input; the summary updates live.

4.3 Requirements Potential Changes

Potential requirement changes for the next deliverable include the following:

- Addition of other heart-rate therapy modes, such as AOOR, VOOR, AAIR and VVIR.
- In addition to the above states, new parameters may be introduced to ensure effective pace-modulation as well as new timing logic.
- Establishing communication between the DCM and the hardware model to ensure information can be relayed between the two components.

4.4 Design Decision Potential Changes

Potential design decision changes for the next deliverable include the following:

- Adjust the State Flow model to accommodate the newly added pacing modes.

- The top-level model will be adjusted to accommodate DCM communication, through introducing an additional subsystem(s).
- The hardware hiding subsystem may be expanded, to ensure additional mappings to new outputs are accounted for.
- Additional states may be added to ensure safety-critical conditions are accounted for with new mode additions, as well as potential checks for signal integrity with the DCM

4.5 Module Description

Table 16: Descriptions of Each Module in DCM

File	Purpose	Key Behavior
main.py	Entry point for the desktop DCM application.	Imports DCMApp from dcm_app.app. Instantiates the root application (app = DCMApp()) and starts the Tkinter main loop via app.run().
dcm_app/app.py	Root Tkinter application and screen manager for the Pacemaker DCM.	When instantiated, DCMApp constructs the main window, allocates a root container for all UI screens, and initializes the StorageService, CommsService, and ValidationService. It manages the authenticated user session, loads or initializes that user's PacemakerSettings, and registers all screen frames for navigation. The application attempts auto-login by restoring the last recorded session;

		otherwise, it displays the authentication screen. All transitions between screens occur through show_frame(), which raises the appropriate view and invokes per-screen lifecycle hooks. Overall, this module orchestrates the core execution flow and ensures that services and UI layers interact coherently.
dcm_app/services/validation_service.py	Performs centralized validation of pacemaker parameter values based on simplified Table-7 physiological and electrical constraints.	The ValidationService receives a PacemakerSettings object and evaluates each parameter against established bounds for rates, amplitudes, pulse widths, sensitivities, refractory periods, and AV timing values. It returns a Boolean validity flag along with a list of descriptive error messages identifying any constraint violations. This ensures that invalid or unsafe parameter configurations are intercepted before they can be saved or transmitted to hardware, serving as a critical safety layer between the user interface and the pacemaker model.
dcm_app/services/storage_service.py	Provides lightweight JSON-based persistence for user	Upon initialization, the service ensures that users.json, settings.json,

	accounts, per-user pacemaker settings, and session state.	and session.json exist and are structurally valid. It supports user registration with SHA-256 password hashing, enforces a maximum user count, validates login credentials, restores previously saved pacemaker settings, and maintains the identity of the currently logged-in user. If any JSON file becomes corrupted or unreadable, the service automatically resets it to a safe default structure to preserve system stability. Through this module, the DCM maintains continuity across sessions without relying on external databases or browser storage.
dcm_app/services/comms_service.py	Implements UART serial communication, parameter encoding/decoding, live egram streaming, and UART traffic logging for hardware interaction.	The CommsService manages the full serial communication lifecycle, from enumerating available COM ports to connecting, disconnecting, and verifying port responsiveness. It encodes PacemakerSettings into a compact 15-byte frame, transmits it to the pacemaker hardware, and decodes incoming frames back into structured settings. All transmitted and received bytes are logged to uart_log.txt with

		timestamps for traceability. The module also supports continuous egram acquisition via a background thread that reads raw serial data and pushes samples into a queue for visualization. Additionally, it includes tools for loopback and round-trip testing to confirm UART reliability between the DCM and the embedded pacemaker model.
dcm_app/models/settings.py	Defines the full data model for programmable pacemaker settings, including bradycardia parameters, refractory periods, timing controls, and rate-responsive features.	This module encapsulates all adjustable pacemaker parameters in a structured @dataclass, providing clear defaults and seamless conversion to and from dictionary form for JSON storage and UART encoding. It acts as the canonical representation of a configuration profile within the DCM, ensuring consistent parameter handling across the UI, validation, storage, and communication layers.
dcm_app/models/egram.py	Represents a single electrogram (egram) measurement used for real-time atrial and ventricular waveform display.	The EgramSample dataclass holds the timestamp, measured voltage, and chamber identifier for each egram point. It provides a simple and lightweight data representation that

		integrates cleanly with the egram queueing and plotting mechanisms implemented in the communication and UI layers.
dcm_app/models/user.py	Defines the data structure for registered DCM users.	The User dataclass stores the username and hashed password of each registered account. Instances of this class are serialized into and deserialized from users.json by the StorageService, enabling user authentication, session restoration, and enforcement of account uniqueness with minimal overhead.

4.6 Testing

Test 1: Basic Registration

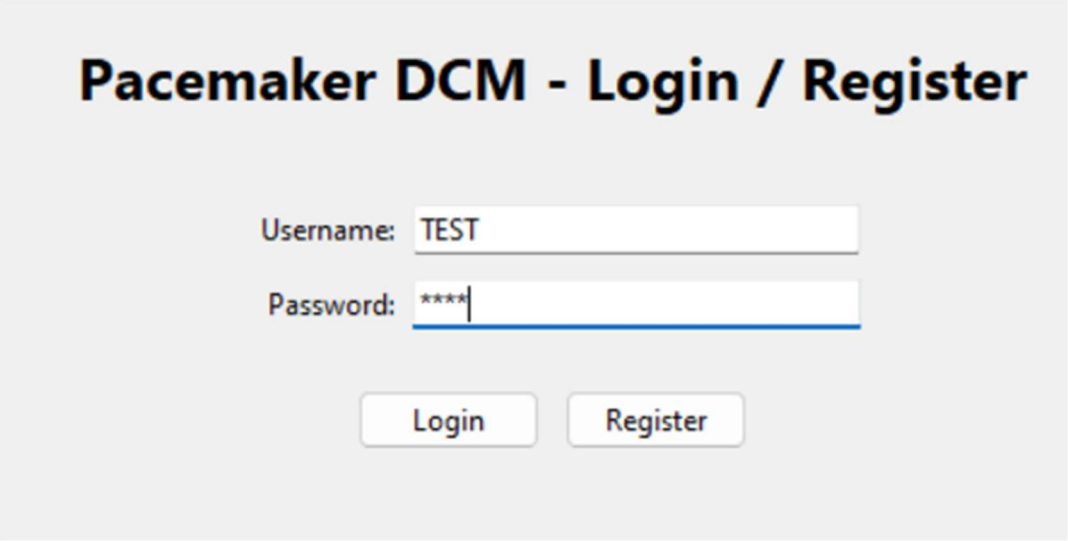


Figure 34: Basic Registration Test Output

Purpose: Verify a new user can be created and that the DCM enforces the 10-user maximum.

Input conditions: Register a new user when the total registered users is still below 10.

Expected output: The system allows the registration and automatically logs in, navigating to the Dashboard.

Actual output: The DCM confirmed registration and immediately navigated to the Dashboard because the total number of users was below the 10-user limit. The interface also enforces a maximum of 10 users, preventing further registrations once the limit is reached.

Result: Pass

Test 2: Dashboard Display

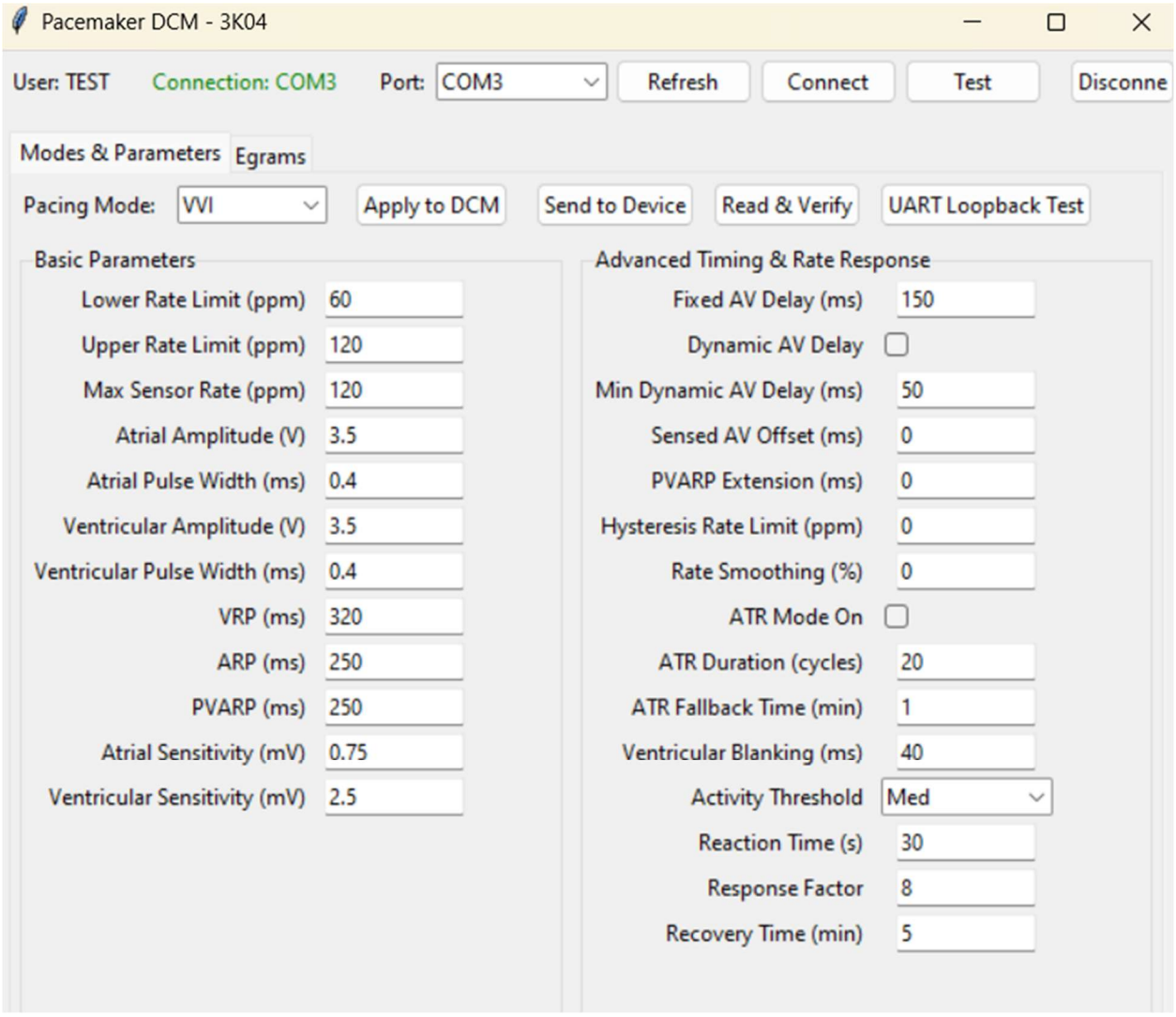


Figure 35: Dashboard Display Test Output

Purpose:

Verify that after registration/login the dashboard correctly displays pacemaker status and

navigation options.

Input conditions:
Register or log in successfully and arrive on the Dashboard.

Expected output:
The Dashboard is displayed showing:

- the logged-in username,
- pacemaker status (Connected / Online),
- the currently active pacing mode (VVI by default),
- battery status,
- system information,
- and navigation tabs for Pacing Modes and Parameters.

Actual output:
After registration, the DCM automatically navigated to the Dashboard and displayed the pacemaker status as **Online**, with VVI shown as the active mode and device/system details visible. Navigation to other modules was available.

Result: Pass

Test 3: Dashboard Offline (Incorrect/Unconnected COM Port)

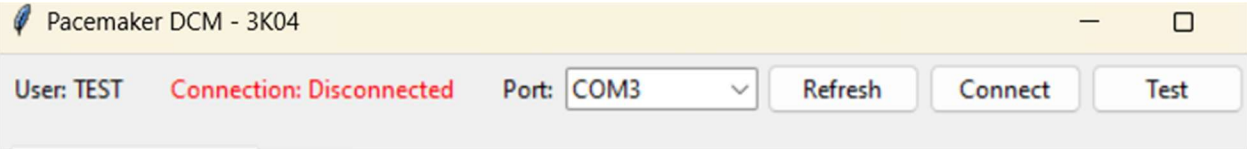


Figure 36: Dashboard Offline Test Output

Purpose:
Verify that the dashboard reflects an *offline* device state when the pacemaker is not connected (e.g., user selected COM8 instead of COM3).

Input conditions:
User is logged in, but either:

- no COM port has been configured yet, **or**
- an invalid / non-working port (COM8) is selected.

Expected output:
The dashboard should display:

- **Device status = Offline (red)**
- **Connection indicator = Not Connected** in the header
- System information remains static (but with no active link to hardware)
- Pacing mode remains visible as last-selected (VVI), but no interaction with device

Actual output:
The dashboard displayed **Offline** status with a red label. The header shows **Not Connected** on COM8, and system details remain visible but inactive. The pacing mode is still shown as VVI, but no connection to hardware is active.

Result: Pass

Test 4: Pacemaker Connection (Successful on COM3)

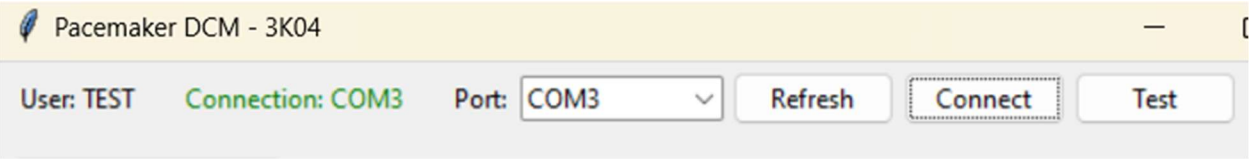


Figure 37: Pacemaker Connection Test Output

Purpose:
Verify that the DCM can successfully detect and display a valid pacemaker connection when the correct COM port is selected.

Input conditions:
User opens the connection modal from the Dashboard, selects **COM3**, and clicks "Test Connection".

Expected output:
The system validates the port and displays a success confirmation:

- "Successfully connected to pacemaker on COM3."
- Connection status turns **Connected (green)**.
- The global header updates to **Connected** and the dashboard reflects **Online** status.

Actual output:
The modal displayed a green success message confirming connection on COM3. After applying, the dashboard updated to **Online** and the header showed **Connected (COM3)**.

Result: Pass

Test 5: Pacing Modes Visibility (Static Active Mode)

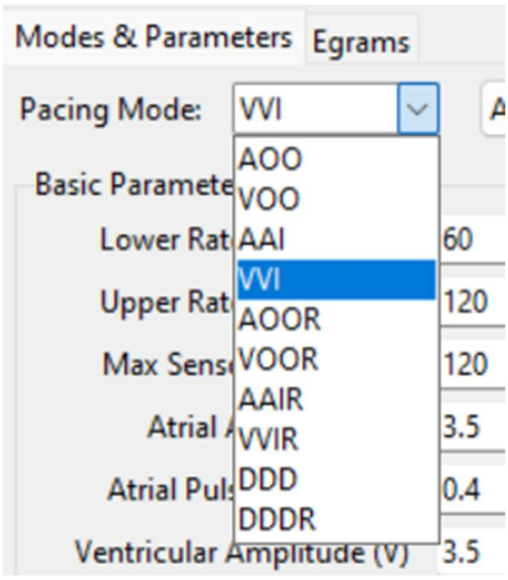


Figure 38: Pacing Modes Visibility Test Output

Purpose:
To verify that all supported pacing modes are displayed to the user and that the currently active mode is clearly indicated both on the Pacing Modes page and on the Dashboard.

Input Conditions:
User successfully logs in and navigates to the **Pacing Modes** screen while the device is connected.

Expected Output:

- All four Deliverable-1 modes (**AOO**, **VOO**, **AAI**, **VVI**) are visible.
- One of the modes (initially **VVI**) is marked as **Active**.
- The Dashboard **Current Mode** section also displays **VVI**, ensuring consistency across UI screens.
- The user can view the details of each mode before switching.

Actual Output:
All supported pacing modes were listed, and **VVI** was clearly displayed with the "Active" badge. The Dashboard also showed **VVI** in the Current Mode card prior to any mode switch, confirming consistency.

Result: Pass

Test 6: Parameters Page: Visibility & Defaults

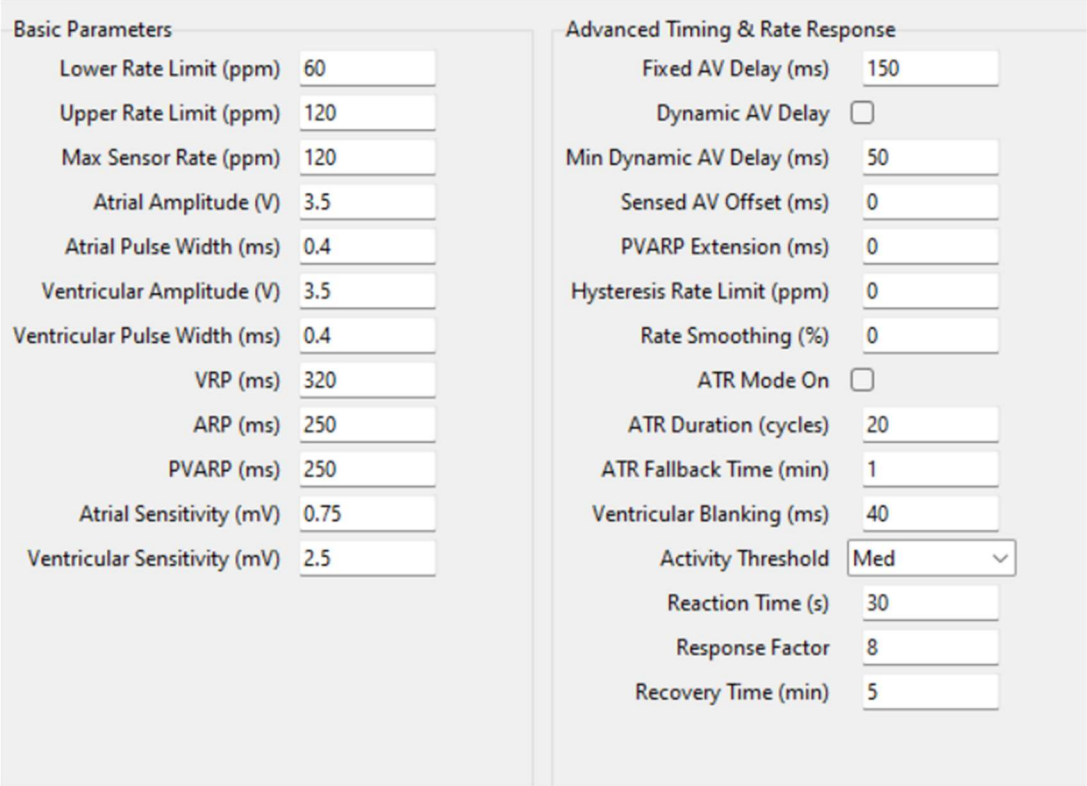


Figure 39: Parameters Page Visibility Test Output

Purpose:
Verify that the Parameters screen loads correctly, shows all required fields, and displays default values with the summary visible.

Input conditions:
From the Dashboard, navigate to **Parameters**.

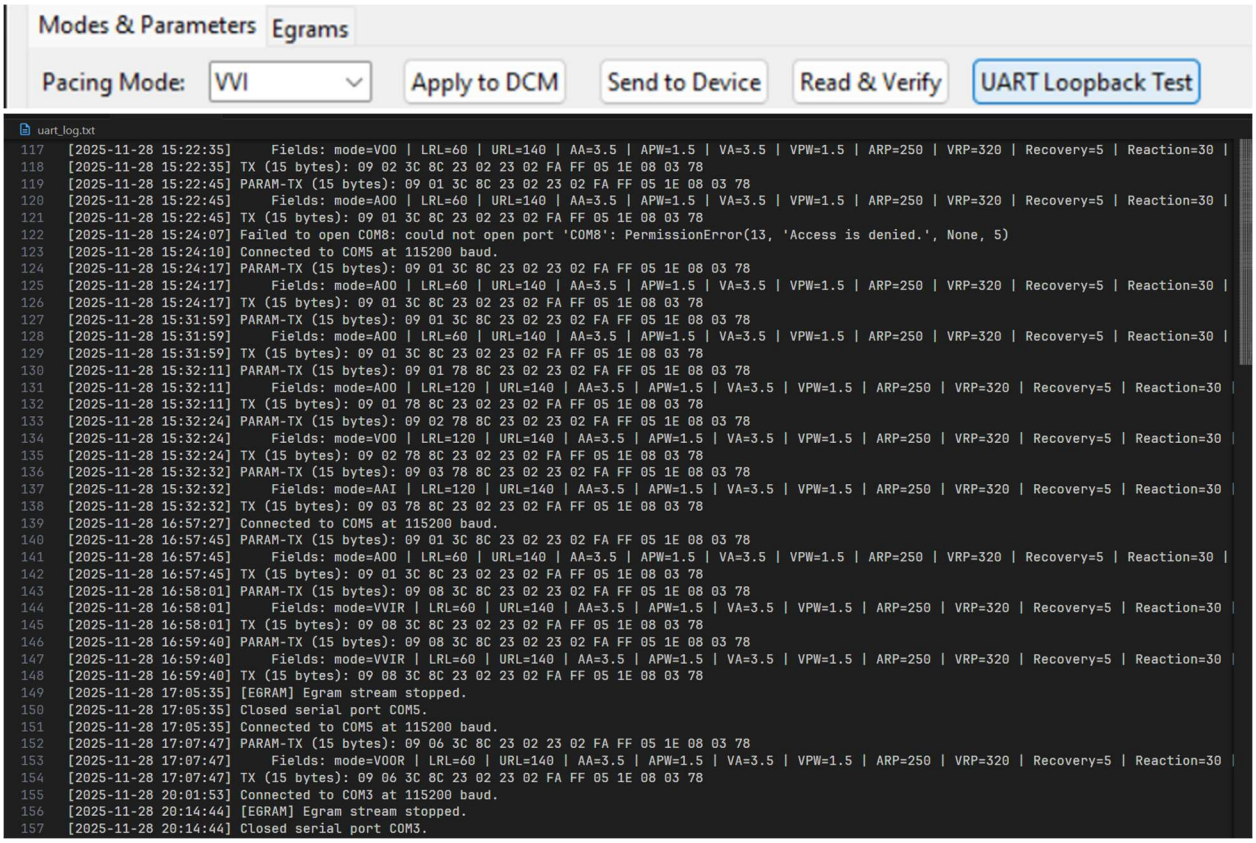
Expected output:

- All eight parameters are visible: **LRL, URL, Atrial Amplitude, Atrial Pulse Width, Ventricular Amplitude, Ventricular Pulse Width, ARP, VRP**.
- Each parameter shows its **default value** and has all three controls available (**slider, +/- stepper, direct numeric field**).
- The **Parameter Summary** section is visible and reflects the current defaults.

Actual output:
All eight parameters were listed with defaults pre-filled. Sliders, steppers, and numeric fields were present for each. The Parameter Summary appeared and matched the defaults.

Result: Pass

Test 7: UART loopback test



Purpose:

Verify that the DCM can send parameter frames over a COM port and that any data echoed or returned on the same port is correctly logged in uart_log.txt.

Input conditions:

Connect the DCM to the pacemaker Simulink model on a valid COM port (for example COM5 or COM3 at 115200 baud), change pacing modes so that multiple parameter frames are sent, and run the system while the model is able to transmit arbitrary data back.

Expected output:

uart_log.txt shows PARAM-TX and TX entries for each frame sent, along with at least one RX entry containing received bytes and a corresponding [PARAM] Decoded from RX: line. The exact RX values are not fixed; any nonempty received data is sufficient to confirm bidirectional communication, even if warnings appear about an unexpected sync byte.

Actual output:

The log contains many PARAM-TX and TX lines for modes such as AOO, VOO, AAI, VVI, VOOR, AOOR, VVIR, and DDDR, followed by multiple RX (15 bytes) entries and [PARAM] Decoded from RX: messages with decoded fields. Connections to COM5 and COM3 at 115200

baud are recorded, and the DCM continues to send, receive, and log data without errors.
Result: Pass

5 Assurance Case

5.1 Requirements-Focused Safety Rationale

The pacemaker developed in this project is argued to be safe for use across all supported pacing modes (AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, VVIR). The safety justification is based on showing that:

1. All identified safety-critical requirements are reflected in the system design, and
2. The system behaves safely both under normal operating conditions and in edge cases.

Evidence collected through verification and testing demonstrates that the safety-related behaviors outlined in the specifications are correctly implemented. Examples include:

- Lower rate limit, upper rate limit, and maximum sensor-controlled pacing limits are implemented exactly as defined.
- In AAIR and VVIR modes, the escape interval and refractory logic are correctly enforced, preventing timing conflicts with natural heart activity.
- AOO and VOO operate strictly in asynchronous mode with no sensing, ensuring predictable pacing at fixed intervals.
- Rate-adaptive modes use reaction and recovery timing rules to keep pacing within a safe physiological window.
- Testing confirms that the device never delivers pulses to unintended chambers.

5.2 Hazard Identification

A Failure Modes and Effects Analysis (FMEA) was conducted to identify potential hazards. The analysis assigns severity (S), occurrence (O), and detection difficulty (D) ratings to compute a Risk Priority Number ($RPN = S \cdot O \cdot D$). Higher RPN values indicate areas requiring greater mitigation effort.

Table 17: FMEA Hazard Analysis

Failure Mode	Cause	Effect on Patient	S	O	D	RPN
Wrong Chamber Paced	Incorrect pin mapping	Hemodynamic instability	9	1	3	27
Failure to Pace	State error in software	Bradycardia	10	3	2	60
Over/Under Pacing	Parameter output error	Arrhythmia	8	2	5	80
Oversensing	Noisy accelerometer input	Arrhythmia	9	4	5	180
Under-sensing	Sensitivity too low	Tachycardia	9	4	7	252

5.3 Hazard Mitigation Strategies

Mitigation actions were proposed or implemented for each identified failure mode:

5.3.1 Failure to Pace

- Continuously monitor for illegal or stalled states and transition into a safe fallback mode when necessary.
- Ensure the Stateflow model is fully deterministic with no unreachable or undefined states.
- Provide a robust reset mechanism that reinitializes all timing counters and states predictably.

5.3.2 Oversensing

- Apply low-pass filtering to accelerometer data so short, high-frequency noise does not produce false activity indications.
- Increase pacing rate only when movement exceeds a defined activity threshold to reduce sensitivity to noise.

5.3.3 Under-sensing

- Allow comparator thresholds to be configurable but within safe, validated bounds.
- Require multiple consistent samples or a minimum pulse width before considering a sensed event valid.

5.3.4 Over/Under Pacing

- Enforce strict parameter validation on the DCM to prevent unsafe amplitude, width, or timing inputs.
- Reject or warn against values outside clinically acceptable ranges.

5.3.5 Wrong Chamber Pacing

- Use fixed, reviewed constants for hardware pin mapping.
- Introduce an intermediate logical “atrial pace” and “ventricular pace” signal layer, where only the final mapping module touches GPIO pins, making testing and verification isolated to one subsystem.

5.4 Mode-Specific Safety Requirements

Each pacing mode has specific safety rules that the system must uphold:

5.4.1 AOO Safety

- No atrial or ventricular sensing is performed.
- Only atrial pulses are delivered at the programmed interval.

5.4.2 VOO Safety

- Ventricular pacing occurs at fixed intervals.
- No sensing influences pacing behavior in this mode.

5.4.3 AAI Safety

- Atrial pacing is delivered only when no intrinsic atrial activity is sensed.
- Intrinsic atrial activity always inhibits pacing.
- No ventricular pacing or sensing occurs.

5.4.4 VVI Safety

- Ventricular pacing is delivered only when no intrinsic ventricular activity is sensed.
- Ventricular sensing inhibits pacing.
- Atrial channels do not affect decisions.

5.4.5 AOOR Safety

- Atrial pacing rate is adjusted based on the sensor-controlled rate.
- Sensor-controlled adjustments are constrained by reaction/recovery times and LRL/MSR limits.

5.4.6 VOOR Safety

- Ventricular pacing rate follows the same sensor-driven rules as AOOR.
- All adjustments remain within safety bounds (LRL to MSR).

5.4.7 AAIR Safety

- If no intrinsic atrial event is sensed, atrial pacing occurs based on the current sensor-controlled escape interval.
- Reaction/recovery time rules and LRL–MSR limits also constrain activity-based rate changes.

5.4.8 VVIR Safety

- Ventricular pacing is driven by the sensor-controlled escape interval when intrinsic ventricular activity is absent.
- All pacing remains within the programmed physiological limits.

6 GenAI Usage

For this project, ChatGPT was the primary tool used to assist with shaping the framework and structure of our Pacemaker DCM website. ChatGPT helped us in organizing the basic layout, structuring key sections, and providing suggestions for resolving minor code issues. While ChatGPT didn’t take over the actual coding and customization, it accelerated the development process by assisting with problem-solving and making some of the technical aspects clear.