3K04 Requirements Specification of the Pacemaker

Team #16: !Heartbreakers

November 29, 2024

Group Members:
Murad Ammar, 400436255
Safana Al-Emara, 400452684
Kevin Huang, 400448168
Nathan Lo, 400442840
Dylan Vaughn Villanueva, 400454912
Tharshigan Vithiyananthan, 400461766

Contents

1 Introduction

Conduction disorders (CDs) disrupt the heart's natural conduction system, potentially causing irregular heart rhythms. CDs can develop due to a range of factors, including heart disease, congenital defects, medication side effects, surgical complications, and more.

Patients with these conditions require a reliable device to monitor and regulate heart activity, effectively managing symptoms associated with bradycardia, arrhythmia, heart failure, and other rhythm disorders.

A cardiac pacemaker is a specialized medical device designed to both monitor and pace the heart. By delivering targeted electrical stimulation to the heart chambers, it maintains an appropriate heart rate, preventing conditions like bradycardia, arrhythmia, and, in some cases, heart failure. As a result, pacemakers can relieve patients of symptoms such as shortness of breath, fatigue, dizziness, and other effects associated with an irregular heartbeat and heart failure.

Modern pacemakers are also highly customizable and programmable, offering personalized pacing parameters and modes tailored to each patient's unique needs. This documentation will explore the following four modes and their corresponding adjustable parameters.

Table 1: Tabular expression of modes.

Chamber	**Action**	**Resulting Function Name**
Ventricle	Pace	VOO
	Pace, Sense & Inhibit	VVI
	Rate Adaptive Pacing	VOOR
	Rate Adaptive Pacing, Sense & Inhibit	VVIR
Atrium	Pace	AOO
	Pace, Sense & Inhibit	AAI
	Rate Adaptive Pacing	AOOR
	Rate Adaptive Pacing, Sense & Inhibit	AIIR

As summarized in the table above, eight pacing modes are explored: VOO, VVI, VOOR, VVIR AOO, AAI, AOOR and AAIR. The VOO mode paces the ventricle without evaluating intrinsic heart activity, delivering pulses at a fixed rate regardless of the heart's own rhythm. The VVI mode monitors the heart's activity and inhibits pacing if the heart is beating regularly, ensuring ventricular pacing only when needed. The VOOR mode paces the ventricle based on the activity of the user without evaluating intrinsic heart activity, delivering pulses at a fixed rate regardless of the heart's own rhythm. The VVIR mode monitors the heart's activity and inhibits pacing if the heart is beating regularly, ensuring ventricular pacing only when

needed at a rate set by the activity of the user. The AOO mode functions similarly to VOO but paces the atrium at a fixed rate, without assessing the natural heart activity. The AOOR mode functions similarly to VOOR but paces the atrium at a rate based on the activity of the user, without assessing the natural heart activity. The AAI mode monitors intrinsic atrial activity and provides pacing only if the atrium's natural rhythm is insufficient, effectively inhibiting pacing when the heart functions normally. Lastly, AAIR mode monitors intrinsic atrial activity and provides pacing only if the atrium's natural rhythm is insufficient, effectively inhibiting pacing at a rate based on the activity of the user when the heart functions normally.

2 Requirements

The following requirements were put in place prior to developing the pacemaker:

1. The pacemaker should be able to switch between multiple pacing modes.
2. The pacemaker must allow the user to program voltage and widths values on the device which outputs electrical stimulation to the heart for pacing.
3. The atrial and ventricular pacing pulse amplitudes must be independently programmable
4. The atrial and ventricular pacing pulse widths must be independently programmable
5. Simulink must be used to develop the permanent AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, and VVIR state flows which will be used to control the various PACEMAKER modes.

3 System Black Box

The Pacemaker System is defined as the Device Controller Monitor (DCM) and the pacemaker (Simulink).

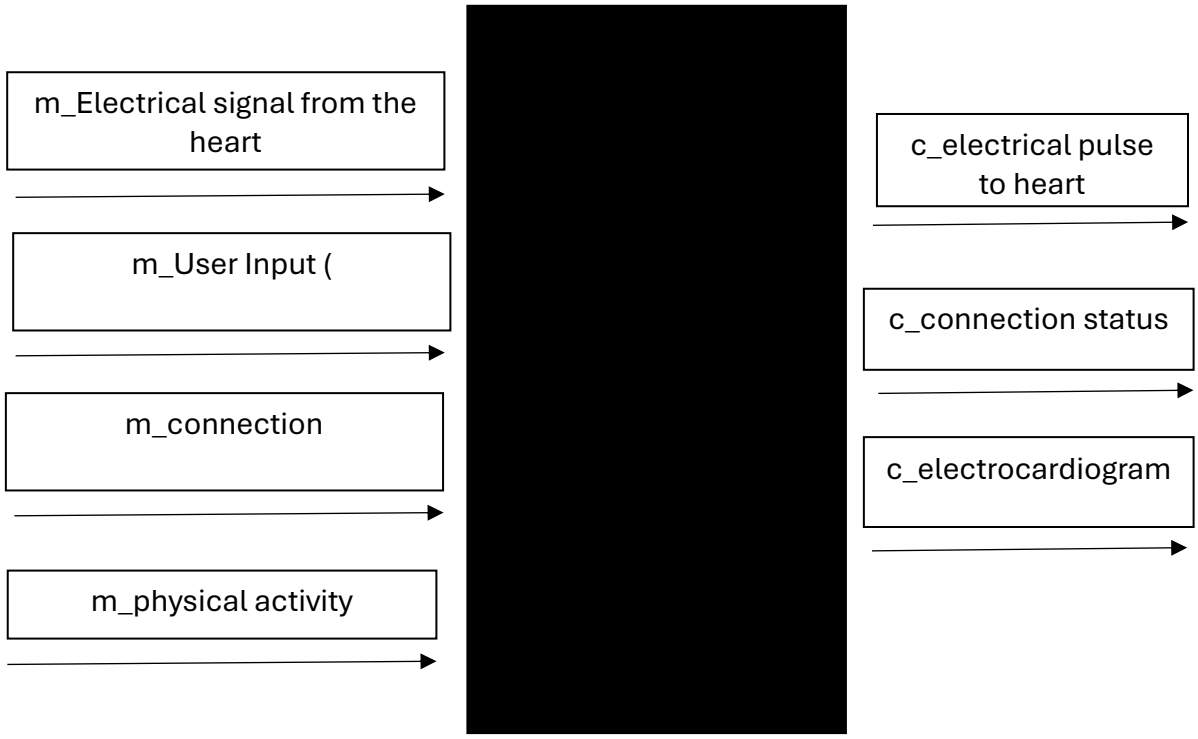


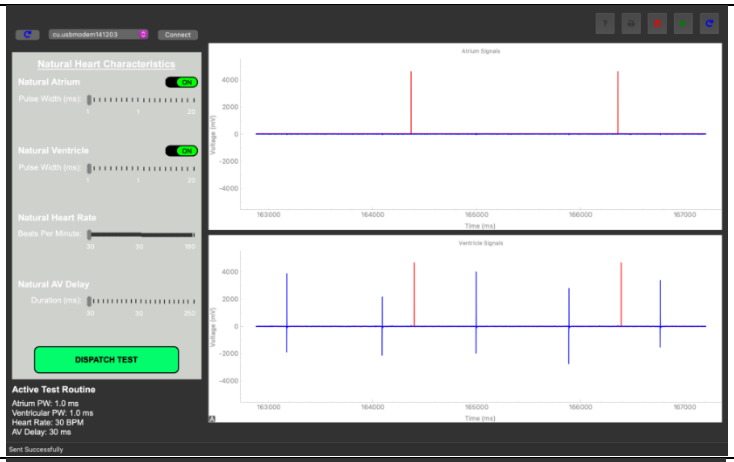
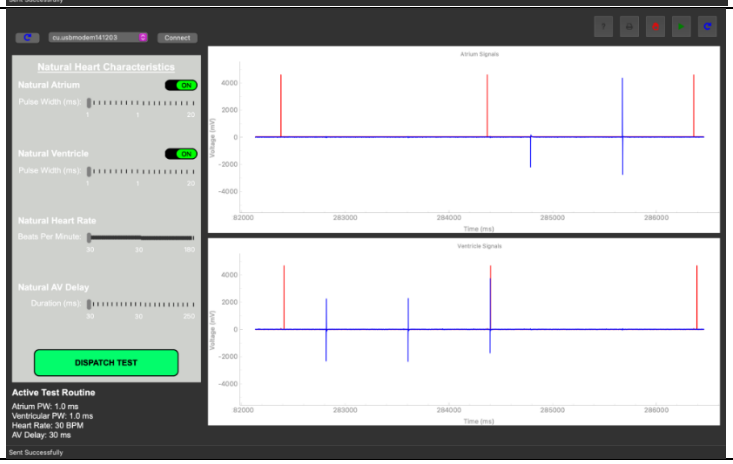
Figure 1: Black Box Diagram for Pacemaker System

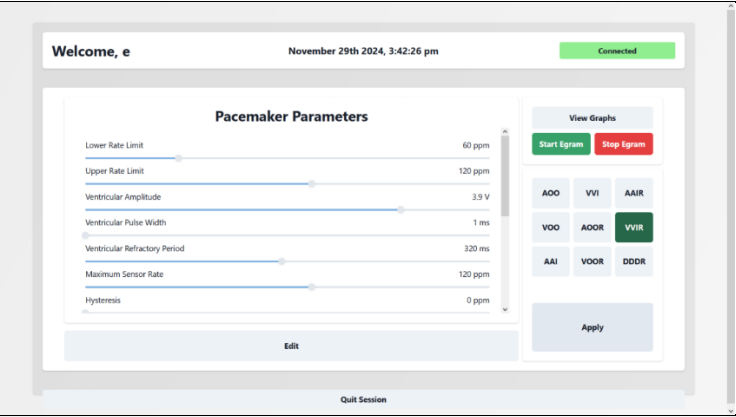
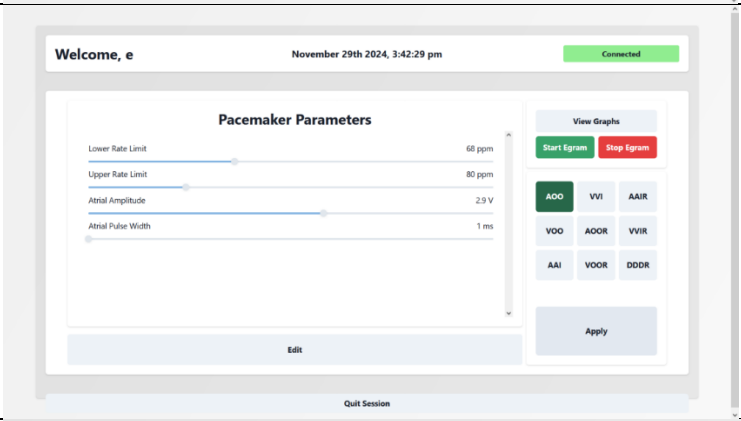
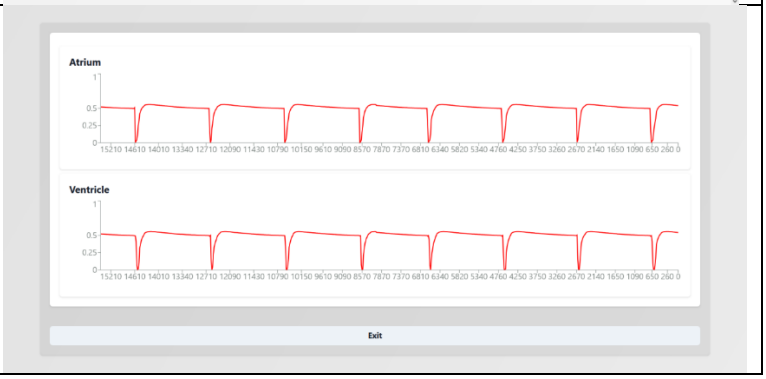
4 System Testing

The system as a whole was tested for functionality and integration. In the test case, the inputs and expected outputs are as follows:

- Inputs
    - the heart was beating at 30 bpm
    - the user input was a change in modes from VVIR to AOO
    - the DCM was physically connected to the pacemaker
    - the pacemaker was shaken to simulate physical activity
- Expected Outputs
    - Electrical pulse from pacemaker at increasing bpm in VVIR
    - Electrical pulse from pacemaker at 60 bpm in AOO
    - “Connected” connection status on the DCM
    - Egram with corresponding bpm

Table 2: The system testing overview.

Actual Outputs	
Heartview in VVIR	
Heartview in AOO	

DCM input for VVIR	
DCM input for AOO	
DCM egram output	

As illustrated above, the actual outputs align with the expected outputs. Therefore, the Pacemaker System behaves as expected.

5 Pacemaker

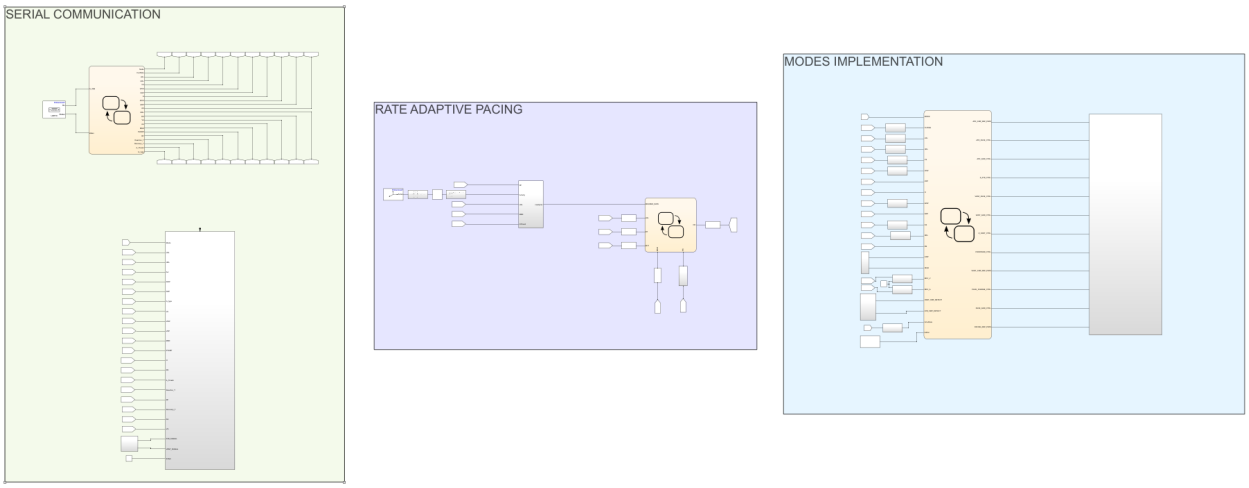


Figure 2: Overall Simulink Model.

5.1 VOO/Default

5.1.1 Description

In VOO mode, the pacemaker paces the ventricle irrespective of the intrinsic heart rate, operating exclusively based on the programmed heart rate. A Stateflow diagram was created in Simulink to model this behavior. The following section provides an overview of the mode's requirements, the constants and parameters used, and a detailed explanation of the Simulink Stateflow implementation.

Given a set heartrate to pace at, when the pacemaker is in VOO mode, then pace the ventricle.

5.1.2 Requirements

1. Must take a user set heartrate to pulse at.
2. Must take a user set pulse width for pulse delivery.
3. Must pace the atrium without sensing or responding to intrinsic heart activities in the ventricle.

5.1.3 Variables and Constants

Figure 3: Black Box Diagram for VOO

5.1.3.1 Monitored Variables

Table 3: Monitored Variables for VOO.

Variable	Name	Range	Nominal Value	Description
Current Rate	CurRate	LRL-URL ppm	-	Current ppm which pulses are being delivered
Ventricular Pulse Width	VPW	0.1-1.9 ms	0.4 ms	Width of the pulse delivered to the ventricle.
Desired Duty Cycle	DDC	0-100 %	$\text{DDC} = \frac{\text{VA}}{\text{Max Volts}} * 100\%$ $\text{DDC} = \frac{3.5\ \text{V}}{5\ \text{V}} * 100\%$ $\text{DDC} = 70\%$	Percentage of time a signal is HIGH in a PWM system.

5.1.3.2 Controlled Variables

Table 4: Controlled Variables for VOO.

Variable	Name	Range	Nominal Value	Description
Pacing Reference Pulse Width Modulation	PACING_REF_PWM	0-100 %	DDC	Percentage of time a signal is HIGH in a PWM system to charge primary capacitor

5.1.3.3 Programmable Parameters

Table 5: Programmable Parameters for VOO.

Variable	Name	Range	Nominal Value	Description

Lower Rate Limit	LRL	50-90 ppm	60 ppm	Minimum ppm at which pulses are delivered in the absence of sensed activity
Upper Rate Limit	URL	50-175 ppm	120 ppm	Maximum ppm at which pulses are delivered in the absence of sense activity
Ventricular Amplitude	VA	3.5–7.0 V	3.5 V	***

5.1.4 Simulink Explanation

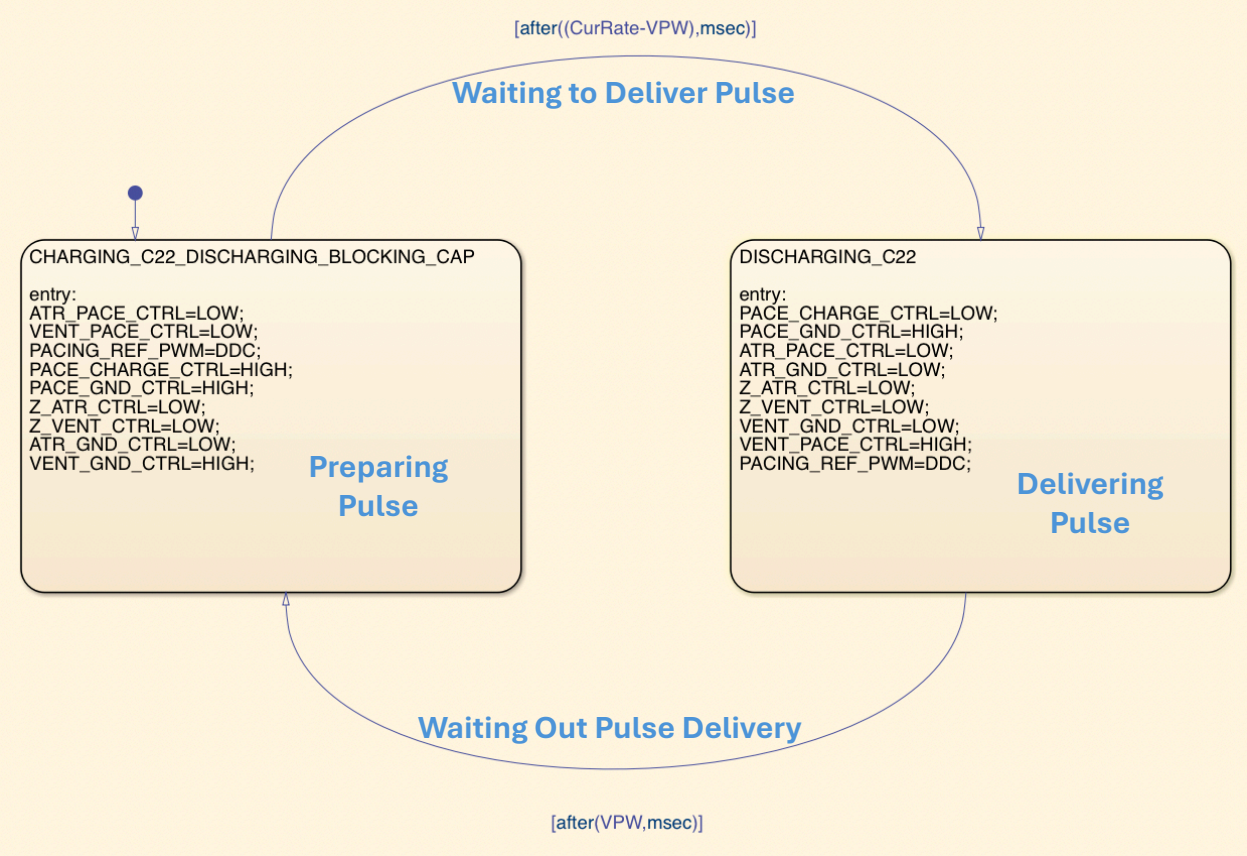


Figure 4: Simulink State Diagram for VOO.

To prepare for pacing the ventricle, the C22 capacitor must first be charged. Afterward, to deliver a pulse, the system must wait for the duration between the ventricular pulse width and the pacing rate interval. Once this interval has passed, the pulse is delivered to the ventricle, and the C22 capacitor discharges into the blocking capacitor. This pulsing event should last precisely for the set pulse width. Following this duration, the C22 pulsing capacitor must be recharged, and the C21 capacitor must be discharged. This process then restarts the pacing cycle.

5.1.4.1 States

Table 6: States for VOO.

State	Description
Charging C22 Discharging Blocking Cap	Charges primary capacitor that will pulse the ventricle.
Discharging C22	Discharges primary capacitor that will pulse the ventricle

Charging C22 Discharging Blocking Cap can be made into two separate states, however since both run at the same time, combined the states into one for simplicity.

5.1.4.2 Variables in States

Table 7: Variables in Charging C22 Discharging Blocking Cap for VOO.

Charging C22 Discharging Blocking Cap	Set Value	Description
ATR_PACE_CTRL	LOW	Used to discharge the primary capacitor into atrium. There is no current flow if LOW.
VENT_PACE_CTRL	LOW	Used to discharge the primary capacitor into ventricle. There is no current flow if LOW.
PACING_REF_PWM	DDC	Used for charging the pacing circuit's primary capacitor.
PACE_CHARGE_CTRL	HIGH	Used to start and stop the charging of the primary capacitor. If HIGH, PWM charges capacitor.
PACE_GND_CTRL	HIGH	Allow current to flow from the ring to the tip in the ventricle.
Z_ATR_CTRL	LOW	Used to examine the electrical connection between the ventricular electrodes and the ventricle itself as well as the ventricular electrode's impedance.
Z_VENT_CTRL	LOW	This switch allows to link the impedance circuit to the ventricle's ring electrode.
ATR_GND_CTRL	LOW	Discharges the electrode’s ring component, or in this case, the blocking capacitor, to reduce charge buildup. If LOW do not discharge the

		blocking capacitor through atrium.
VENT_GND_CTRL	HIGH	Discharges the electrode's ring component, or in this case, the blocking capacitor, to reduce charge buildup. If HIGH, discharge blocking capacitor through ventricle.

Table 8: Variables in Discharging C22 for VOO.

Discharging C22	**Set Value**	**Description**
PACE_CHARGE_CTRL	LOW	Used to start and stop the charging of the primary capacitor. If LOW, PWM disconnected from circuit.
PACE_GND_CTRL	HIGH	Allow current to flow from the ring to the tip in the ventricle.
ATR_PACE_CTRL	LOW	Used to discharge the primary capacitor into atrium. If the switch is set to HIGH, current passes through it.
ATR_GND_CTRL	LOW	Discharges the electrode's ring component, or in this case, the blocking capacitor, to reduce charge buildup. If LOW do not discharge the blocking capacitor through atrium.
Z_ATR_CTRL	LOW	Used to examine the electrical connection between the ventricular electrodes and the ventricle itself as well as the ventricular electrode's impedance.
Z_VENT_CTRL	LOW	This switch allows to link the impedance circuit to the ventricle's ring electrode.
VENT_GND_CTRL	LOW	Discharges the electrode's ring component, or in this case, the blocking capacitor, to reduce charge buildup. If LOW do not discharge blocking capacitor through ventricle.

VENT_PACE_CTRL	HIGH	Used to discharge the primary capacitor into ventricle. If the switch is set to HIGH, current passes through it.
PACING_REF_PWM	DDC	Used for charging the pacing circuit's primary capacitor.

5.1.4.3 Transitions

Table 9: Transitions for VOO.

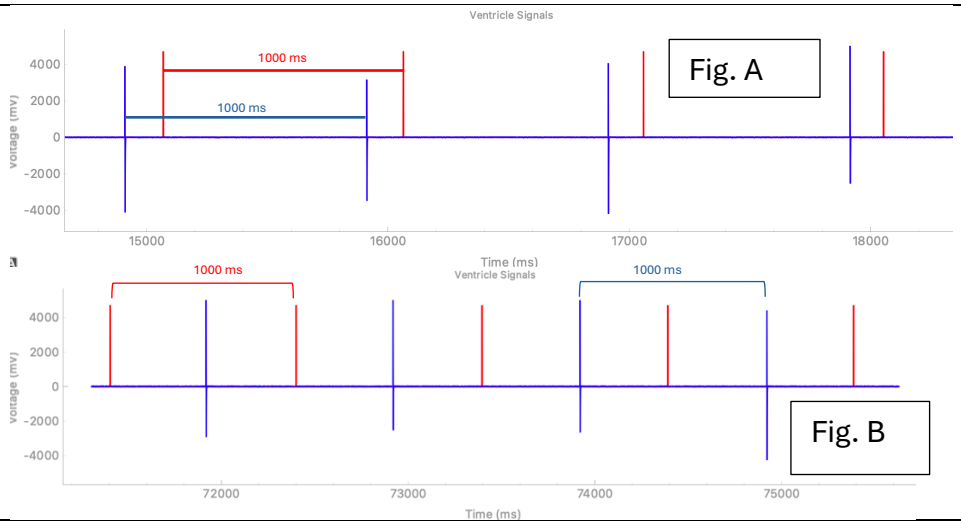
Transition	Description
[after(VPW,msec)]	Transition after waiting VPW milliseconds.
[after((CurRate-VPW),msec)]	Transition after waiting the difference between CurRate and VPW in milliseconds.

5.1.5 Design Justification

Charging the capacitor prior to pulse delivery ensures that sufficient energy is available to effectively pace the heart when needed. Discharging the pulse into another capacitor allows the pacemaker to safely manage and ground the excess charge, preventing accumulation that could lead to irregular performance. This controlled discharge process ensures electrical stability and safety. By setting the interval between pulse preparation and delivery to match the difference between the pacing rate and the ventricular pulse width, the pacemaker optimizes timing, avoiding unnecessary delays. Additionally, waiting the ventricular pulse width duration after delivering the pulse mirrors atrium's natural activity, ensuring efficient and reliable pacing.

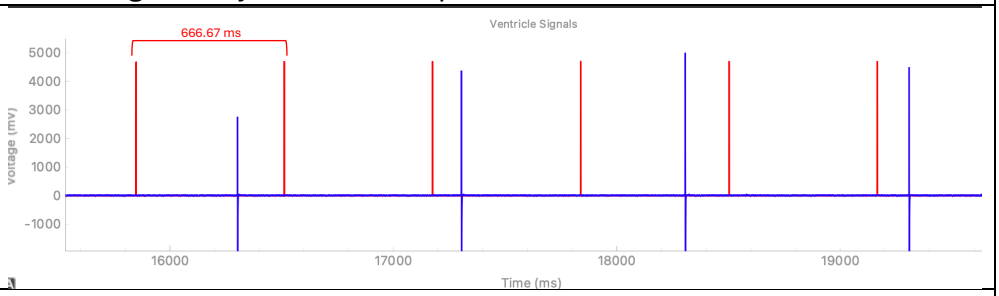
5.1.6 Testing

Table 10: Test Cases and Results for VOO.

Test Case	Result
CurrRate = 60 ppm **VPW = 0.4 ms** **DDC = 70%** **HR = 60 ppm**	
DATUM case	

In Figures A, the pacemaker's pacing interval closely matches the heart's, suggesting similar pulsing rates. However, a slight time shift between the heart and pacemaker pulses indicates a minor interval difference, likely due to the brief charging and discharging of the capacitor processing time. This lag may gradually shift the signals out of phase as it accumulates. The shift is clearly showcased in Figure B of this test case. The pulse width difference between the heart and pacemaker isn't a factor, as the pacemaker's interval timing already accounts for pulse width.

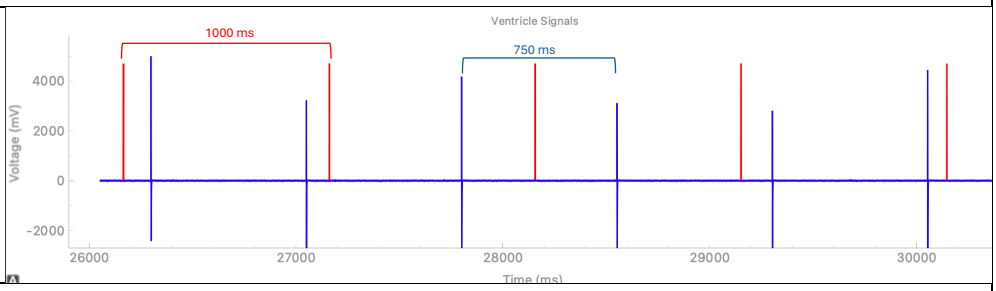
CurrRate = 60 ppm
VPW = 0.4 ms
DDC = 70%
HR = 90 ppm



Test to show no sensing

The image shows that a change in heart rate does not affect the pacemaker's pulsing. In the example, the heart is beating at 90 pulses per minute (ppm), which corresponds to a pulsing interval of 666.67 ms. However, as this mode operates independently of heart activity, its pulse interval remains steady at 1000 ms, unaffected by the heart rate change.

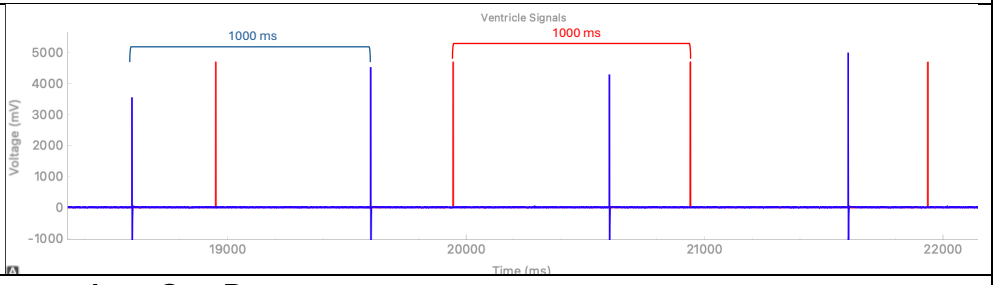
CurrRate = 80 ppm
VPW = 0.4 ms
DDC = 70%
HR = 60 ppm



Test to show increased set CurrRate

The image illustrates that the pacemaker's pulse interval is approximately 750 ms, translating to 80 ppm, which aligns with the simulation's set parameter. Meanwhile, the heart's pulse interval is 1000 ms (60 ppm). This test demonstrates that the VOO mode can operate at a higher rate independently of the heart rate, maintaining its set pacing regardless of intrinsic heart activity.

CurrRate = 40 ppm
VPW = 0.4 ms
DDC = 70%
HR = 60 ppm

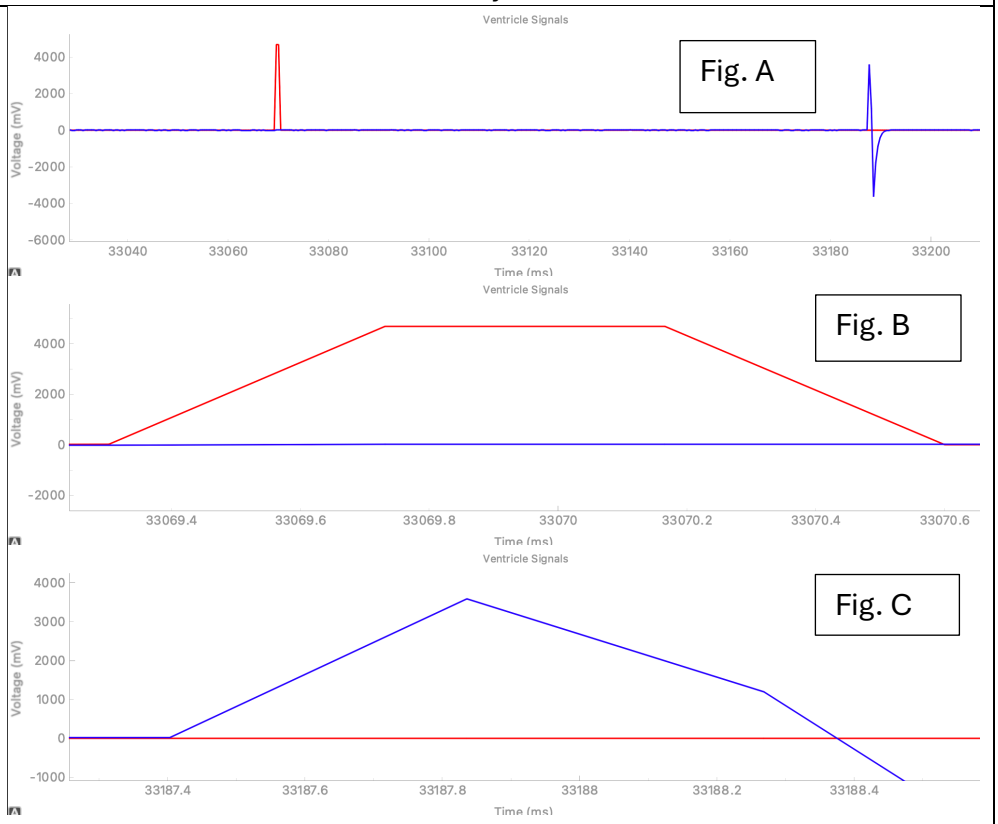


Test to show decreased set CurrRate

The image shows that although the pacemaker's current rate is set to 40 ppm (1500 ms intervals), the actual pulse intervals are around 1000 ms, corresponding to 60 ppm. This

is because the pacemaker's lower rate limit is set to 60 ppm, which overrides the lower setting to prevent bradycardia. The heart rate, also set at 60 ppm, is confirmed by the 1000 ms interval between red pulses. This test demonstrates that in VOO mode, the pacemaker respects the lower rate limit as a boundary it will not fall below.

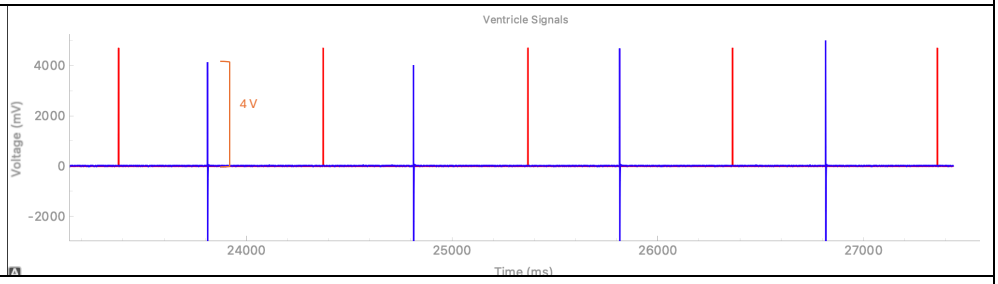
CurrRate = 60 ppm
VPW = 1 ms
DDC = 70%
HR = 60 ppm



Test to show increased set VPW

In Figure A, the heart rate pulse is shown in red and the pacemaker pulse in blue. Figure B, a close-up of the heart rate pulse width, shows a discrepancy: although set to 1 ms in the simulation, it appears closer to 1.3 ms in Heartview. Figure C shows the pacemaker pulse width slightly below the set value of 1 ms, measuring approximately 0.9 ms. These minor discrepancies in pulse width could contribute to the phase lag observed between the natural heart rate and pacemaker pulses. Further testing is required to pinpoint the exact cause.

CurrRate = 60 ppm
VPW = 0.4 ms
DDC = 100%
HR = 60 ppm



Test to show increased set DDC

The DDC setting, ideally at 100%, should result in each pulse reaching a consistent amplitude of 5 V. However, the observed pulse amplitude varies and does not maintain a

steady 5 V throughout the simulations. This inconsistency is likely due to the real-world behavior of the capacitor, where actual charging and discharging patterns deviate from ideal conditions, indicating the capacitor may not be functioning in an ideal manner.

5.1.7 Validation and Verification

This testing confirms that the VOO mode operates independently of the atrium's intrinsic activity and can function at higher pacing rates. Additionally, the VOO mode adheres to the lower and upper heart rate limits. However, inconsistencies were observed between the set DDC and atrial pulse amplitude compared to the values shown in the charts. These discrepancies may result from the non-ideal behavior of the equipment. Further testing is required to pinpoint the exact cause of these variations.

5.1.8 Requirement Changes

As of October 25, 2024, no requirement changes must be made. This is subject to change.

5.1.9 Design Decision Changes

1. Create safety checks for the following:
    a. Inputted Ventricular Pulse Width should be within range of 0.1-1.9 ms.
    b. Inputted Ventricular Amplitude must be limited between 3.5–7.0 V.

5.2 AOO

5.2.1 Description

In AOO mode, given a set heartrate, the pacemaker device paces the atrium without sensing or responding to intrinsic heart activates in the atrium. A state flow diagram was developed in Simulink for this module. The following parameters are variables and inputs into this module.
Given a set heartrate to pace at, when the pacemaker is in AOO mode, then pace the atrium.

5.2.2 Requirements

1. Must take a user set heartrate to pulse to
2. Must take a user set pulse width
3. Must pace the atrium without sensing or responding to intrinsic heart activities in the atrium

5.2.3 Variables and Constants

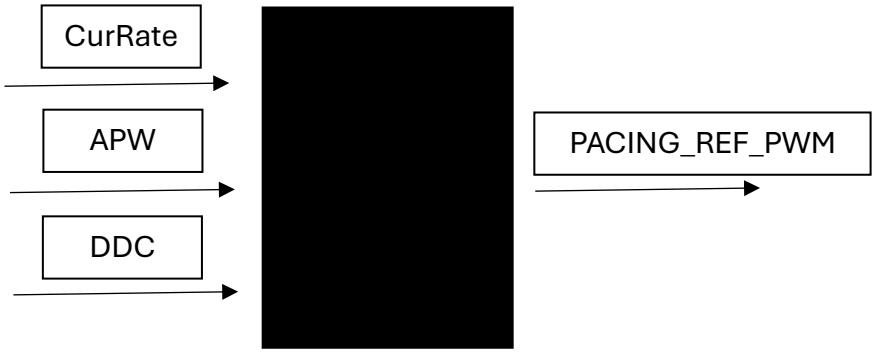


Figure 5: Black Box Diagram for AOO

5.2.3.1 Monitored Variables

Table 11: Monitored Variables for AOO

Variable	Name	Range	Nominal Value	Description
Current Rate	CurRate	LRL-URL ppm	-	Current ppm which pulses are being delivered
Atrial Pulse Width	APW	0.1-1.9 ms	0.4 ms	Width of the pulse delivered to the atrium
Desired Duty Cycle	DDC	0-100 %	$\text{DDC} = \frac{\text{AA}}{\text{Max Volts}} * 100\%$	Percentage of time a signal is HIGH in a PWM system

			$\text{DDC} = \frac{3.5\ \text{V}}{5\ \text{V}} * 100\%$ $\text{DDC} = 70\%$	

5.2.3.2 Controlled Variables

Table 12: Controlled Variables for AOO

Variable	Name	Range	Nominal Value	Description
Pacing Reference Pulse Width Modulation	PACING_REF_PWM	0-100 %	DDC	Percentage of time a signal is HIGH in a PWM system to charge primary capacitor

5.2.3.3 Programmable Parameters

Table 13: Programmable Parameters for AOO

Variable	Name	Range	Nominal Value	Description
Lower Rate Limit	LRL	50-90 ppm	60 ppm	Minimum ppm at which pulses are delivered in the absence of sensed activity
Upper Rate Limit	URL	50-175 ppm	120 ppm	Maximum ppm at which pulses are delivered in the absence of sense activity
Atrial Amplitude	AA	3.5–7.0 V	3.5 V	Amplitude of the voltage of the given pace to the atrium

5.2.4 Simulink Explanation

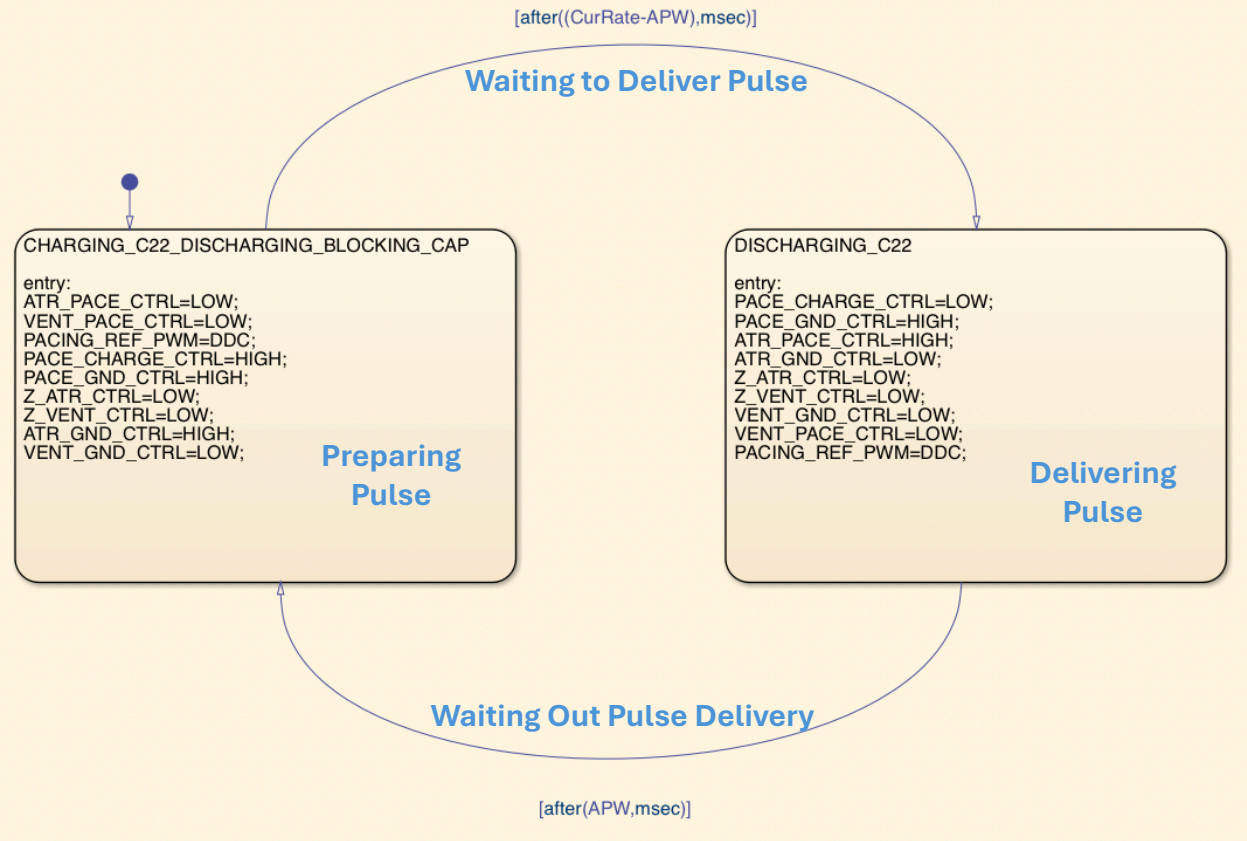


Figure 6: Simulink State Diagram for AOO

To prepare for pacing the atrium, the C22 capacitor must first be charged. Afterward, to deliver a pulse, the system must wait for the duration between the atrial pulse width and pacing rate interval. Once this interval has passed, the pulse is delivered to the atrium, and the C22 capacitor discharges into the blocking capacitor. This pulsing event should last precisely for the set atrial pulse width. Following this duration, the C22 pulsing capacitor must be recharged and the C21 capacitor must be discharged. This will restart the pacing cycle.

5.2.4.1 States

Table 14: States for AOO

State	Description
Charging C22 Discharging Blocking Cap	Charges primary capacitor that will pulse the atrium
Discharging C22	Discharges primary capacitor that will pulse the atrium

Charging C22 Discharging Blocking Cap can be made into two separate states, however since both run at the same time, combined the states into one for simplicity.

5.2.4.2 Variables in States

Table 15: Variables in Charging C22 Discharging Blocking Cap for AOO.

Charging C22 Discharging Blocking Cap	Set Value	Description
ATR_PACE_CTRL	LOW	Used to discharge the primary capacitor into atrium. There is no current flow if LOW.
VENT_PACE_CTRL	LOW	Used to discharge the primary capacitor into ventricle. There is no current flow if LOW.
PACING_REF_PWM	DDC	Used for charging the pacing circuit's primary capacitor.
PACE_CHARGE_CTRL	HIGH	Used to start and stop the charging of the primary capacitor. If HIGH, PWM charges capacitor.
PACE_GND_CTRL	HIGH	Allow current to flow from the ring to the tip in the atrium.
Z_ATR_CTRL	LOW	Used to examine the electrical connection between the atrial electrodes and the atrium itself as well as the atrial electrode's impedance.
Z_VENT_CTRL	LOW	This switch allows to link the impedance circuit to the ventricle's ring electrode.
ATR_GND_CTRL	HIGH	Discharges the electrode’s ring component, or in this case, the blocking capacitor, to reduce charge buildup. If HIGH, discharge blocking capacitor through atrium.
VENT_GND_CTRL	LOW	Discharges the electrode’s ring component, or in this case, the blocking capacitor, to reduce charge buildup. If LOW do not discharge blocking capacitor through ventricle.

Table 16: Variables in Discharging C22 for AOO.

Discharging C22	Set Value	Description
PACE_CHARGE_CTRL	LOW	Used to start and stop the charging of the primary capacitor. If LOW, PWM disconnected from circuit.
PACE_GND_CTRL	HIGH	Allow current to flow from the ring to the tip in the atrium.
ATR_PACE_CTRL	HIGH	Used to discharge the primary capacitor into atrium.

		If the switch is set to HIGH, current passes through it.
ATR_GND_CTRL	LOW	Discharges the electrode's ring component, or in this case, the blocking capacitor, to reduce charge buildup. If LOW do not discharge the blocking capacitor through atrium.
Z_ATR_CTRL	LOW	Used to examine the electrical connection between the atrial electrodes and the atrium itself as well as the atrial electrode's impedance.
Z_VENT_CTRL	LOW	This switch allows to link the impedance circuit to the ventricle's ring electrode.
VENT_GND_CTRL	LOW	Discharges the electrode's ring component, or in this case, the blocking capacitor, to reduce charge buildup. If LOW do not discharge blocking capacitor through ventricle.
VENT_PACE_CTRL	LOW	Used to discharge the primary capacitor into ventricle. There is no current flow if LOW.
PACING_REF_PWM	DDC	Used for charging the pacing circuit's primary capacitor.

5.2.4.3 Transitions

Table 17: Transitions for AOO.

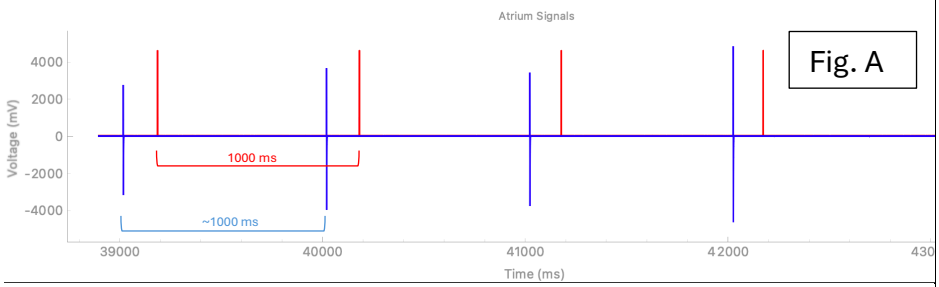
Transition	Description
[after(APW,msec)]	Transition after waiting APW milliseconds
[after((CurRate-APW),msec)]	Transition after waiting the difference between CurRate and APW in milliseconds

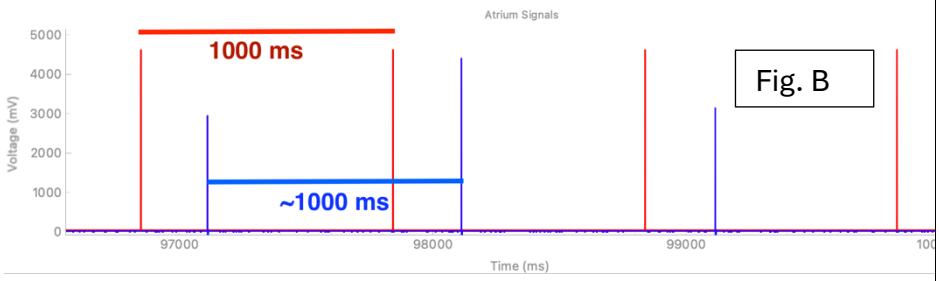
5.2.5 Design Justification

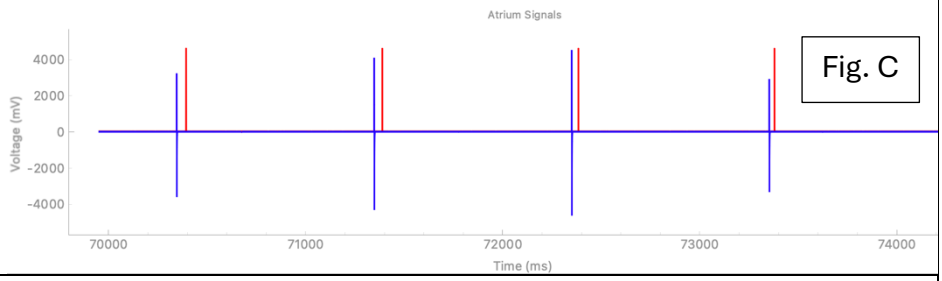
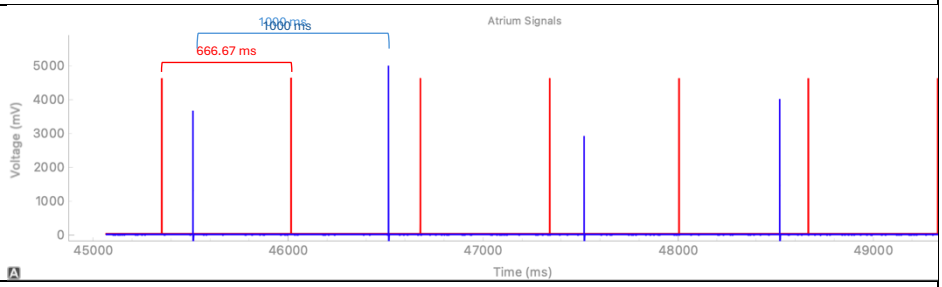
Charging the capacitor prior to pulse delivery ensures that sufficient energy is available to effectively pace the heart when needed. Discharging the pulse into another capacitor allows the pacemaker to safely manage and ground the excess charge, preventing accumulation that could lead to irregular performance. This controlled discharge process ensures electrical stability and safety. By setting the interval between pulse preparation and delivery to match the difference between the pacing rate and the atrial pulse width, the pacemaker optimizes timing, avoiding unnecessary delays. Additionally, waiting the atrial pulse width duration after delivering the pulse mirrors atrium's natural activity, ensuring efficient and reliable pacing.

5.2.6 Testing

Table 18: Test Cases and Results for AOO.

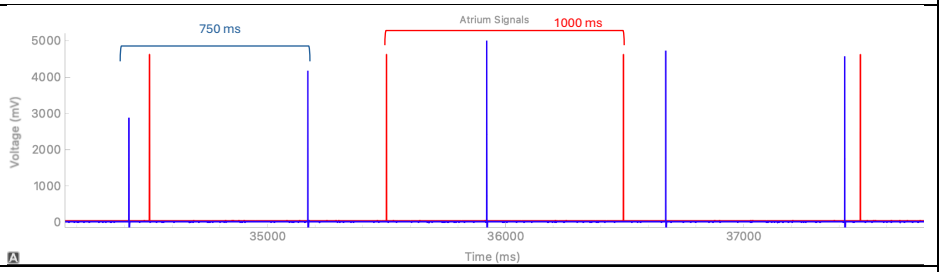
<table>
<tr><th>Test Case</th><th>Result</th></tr>
<tr><td>CurrRate = 60 ppm
APW = 0.4 ms
DDC = 70%
HR = 60 ppm</td><td> 




</td></tr>
<tr><td colspan="2">DATUM case
In Figures A and B, the pacemaker’s pacing interval (shown in blue) closely matches the heart's (shown in red), suggesting similar pulsing rates. However, a slight time shift between the heart and pacemaker pulses indicates a minor interval difference, likely due to the brief charging and discharging of the capacitor processing time. This lag may gradually shift the signals out of phase as it accumulates. The shift is clearly showcased in Figure C of this test case. The pulse width difference between the heart and pacemaker isn’t a factor, as the pacemaker’s interval timing already accounts for pulse width.</td></tr>
<tr><td>CurrRate = 60 ppm
APW = 0.4 ms
DDC = 70%
HR = 90 ppm</td><td> 
</td></tr>
<tr><td colspan="2">Test to show no sensing
The image shows that a change in heart rate does not affect the pacemaker’s pulsing. In the example, the heart is beating at 90 pulses per minute (ppm), which corresponds to a</td></tr>
</table>

pulsing interval of 666.67 ms. However, as this mode operates independently of heart activity, its pulse interval remains steady at 1000 ms, unaffected by the heart rate change.

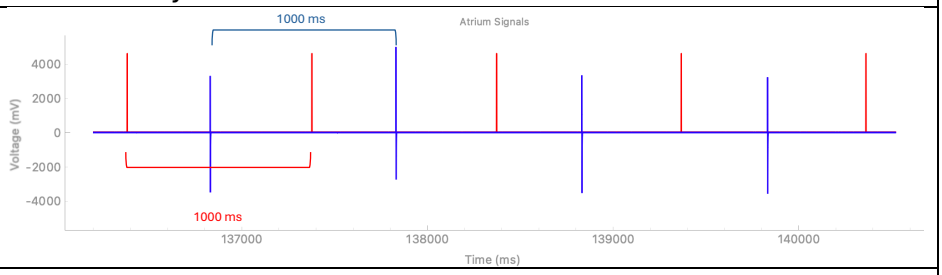
CurrRate = 80 ppm
APW = 0.4 ms
DDC = 70%
HR = 60 ppm



Test to show increased set CurrRate

The image illustrates that the pacemaker's pulse interval is approximately 750 ms, translating to 80 ppm, which aligns with the simulation's set parameter. Meanwhile, the heart's pulse interval is 1000 ms (60 ppm). This test demonstrates that the AOO mode can operate at a higher rate independently of the heart rate, maintaining its set pacing regardless of intrinsic heart activity.

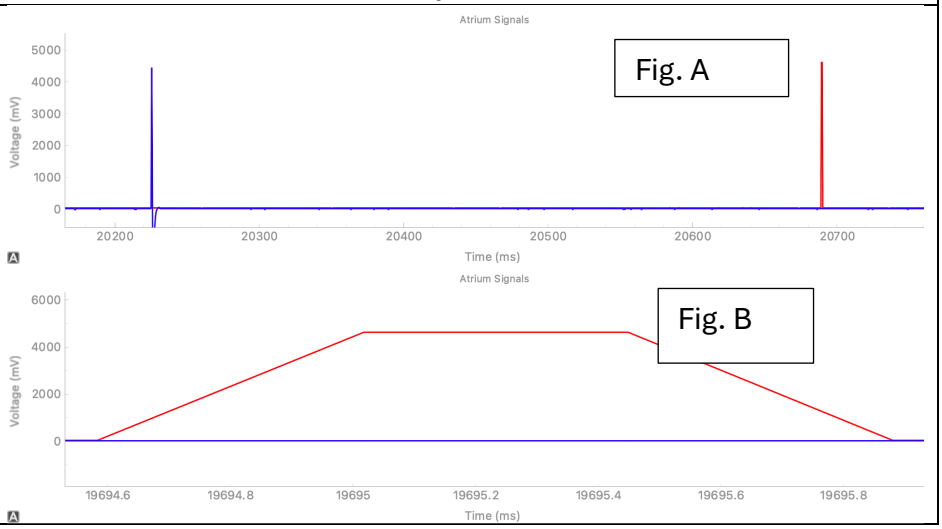
CurrRate = 40 ppm
APW = 0.4 ms
DDC = 70%
HR = 60 ppm

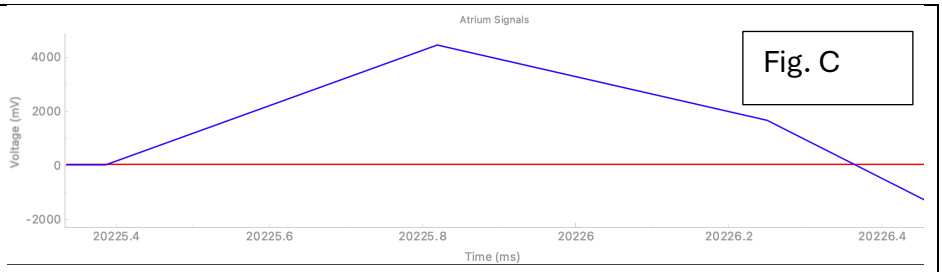


Test to show decreased set CurrRate

The image shows that although the pacemaker's current rate is set to 40 ppm (1500 ms intervals), the actual pulse intervals are around 1000 ms, corresponding to 60 ppm. This is because the pacemaker's lower rate limit is set to 60 ppm, which overrides the lower setting to prevent bradycardia. The heart rate, also set at 60 ppm, is confirmed by the 1000 ms interval between red pulses. This test demonstrates that in AOO mode, the pacemaker respects the lower rate limit as a boundary it will not fall below.

CurrRate = 60 ppm
APW = 1 ms
DDC = 70%
HR = 60 ppm

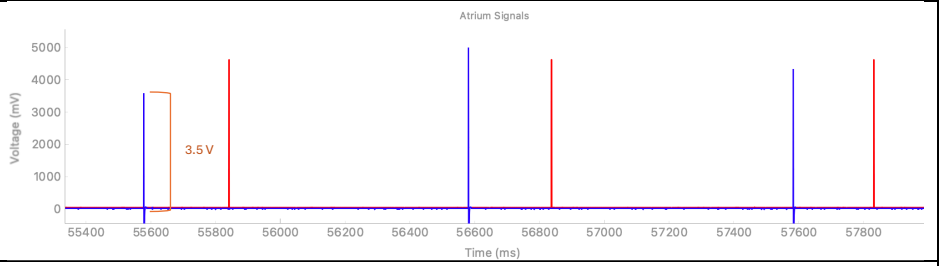




Test to show increased set APW

In Figure A, the heart rate pulse is shown in red and the pacemaker pulse in blue. Figure B, a close-up of the heart rate pulse width, shows a discrepancy: although set to 1 ms in the simulation, it appears closer to 1.3 ms in Heartview. Figure C shows the pacemaker pulse width slightly below the set value of 1 ms, measuring approximately 0.9 ms. These minor discrepancies in pulse width could contribute to the phase lag observed between the natural heart rate and pacemaker pulses. Further testing is required to pinpoint the exact cause.

CurrRate = 60 ppm
APW = 0.4 ms
DDC = 100%
HR = 60 ppm



Test to show increased set DDC

The DDC setting, ideally at 100%, should result in each pulse reaching a consistent amplitude of 5 V. However, the observed pulse amplitude varies and does not maintain a steady 5 V throughout the simulations. This inconsistency is likely due to the real-world behavior of the capacitor, where actual charging and discharging patterns deviate from ideal conditions, indicating the capacitor may not be functioning in an ideal manner.

5.2.7 Validation and Verification

This testing confirms that the AOO mode operates independently of the atrium's intrinsic activity and can function at higher pacing rates. Additionally, the AOO mode adheres to the lower and upper heart rate limits. However, inconsistencies were observed between the set DDC and atrial pulse amplitude compared to the values shown in the charts. These discrepancies may result from the non-ideal behavior of the equipment. Further testing is required to pinpoint the exact cause of these variations.

5.2.8 Requirement Changes

As of October 25, 2024, no requirement changes must be made. This is subject to change.

5.2.9 Design Decision Changes

1. Create safety checks for the following:
    a. Inputted Atrial Pulse Width should be within range of 0.1-1.9 ms.
    b. Inputted Atrial Amplitude must be limited between 3.5–7.0 V.

5.3 VVI

5.3.1 Description

In VVI mode, the pacemaker device paces and senses the ventricle. Detection of a heartbeat above a specified threshold inhibits artificial pacing. A state flow diagram was developed in Simulink for this module. The following parameters are variables that are input into this module.

5.3.2 Requirements

1. Must take a user set heart rate to pulse at
2. Must take a user set pulse width, amplitude, lower rate limit, upper rate limit, hysteresis mode, sensitivity, refractory period, and rate smoothing condition
3. Must not pulse the ventricle when it beats naturally above the current rate
4. Must pulse the ventricle when it beats below the current rate
5. Must not sense during the ventricular refractory period
6. Must allow for hysteresis upon natural ventricular events when hysteresis is active

5.3.3 Variables and Constants

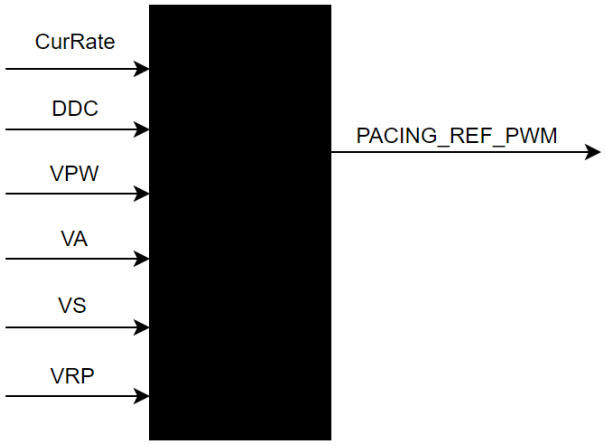


Figure 7: Black Box Diagram for VVI

5.3.3.1 Monitored Variables

Table 19: Monitored Variables for VVI.

Variable	Name	Range	Nominal Value	Description
Current Rate	CurRate	LRL-URL ppm	-	Current ppm which pulses are being delivered
Ventricular Pulse Width	VPW	0.1-1.9 ms	0.4 ms	Width of the pulse delivered to the ventricle

Desired Duty Cycle	DDC	0-100 %	$\text{DDC} = \frac{\text{VA}}{\text{Max Volts}} * 100\%$ $\text{DDC} = \frac{3.5\ \text{V}}{5\ \text{V}} * 100\%$ $\text{DDC} = 70\%$	Percentage of time a signal is HIGH in a PWM system

5.3.3.2 Programmable Parameters

Table 20: Programmable Parameters for VVI.

Variable	Name	Range	Nominal Value	Description
Lower Rate Limit	LRL	50-90 ppm	60 ppm	Minimum ppm at which pulses are delivered in the absence of sensed activity
Upper Rate Limit	URL	50-175 ppm	120 ppm	Maximum ppm at which pulses are delivered in the absence of sense activity
Ventricular Amplitude	VA	3.5–7.0 V	3.5 V	Amplitude of the signal sent to the ventricle
Ventricular Sensitivity	VS	1.0-10 mV	0.5 mV	Voltage level at which to classify sensed signals as heart pulses
Ventricular Refractory Period	VRP	150-500 ms	320 ms	Period following a natural or artificial pulse where sensing is disabled
Hysteresis	H	HIGH or LOW	-	HIGH if hysteresis pacing is enabled. LOW if hysteresis pacing is disabled
Hysteresis Rate Limit	HRL	[LRL, CurRate]	-	Lowest ppm at which hysteresis may inhibit artificial pacing
Rate Smoothing	RS	Off, 3, 6, 9, 12, 15, 18, 21	-	Maximum allowable change in heartrate in percentage

5.3.4 Simulink Explanation

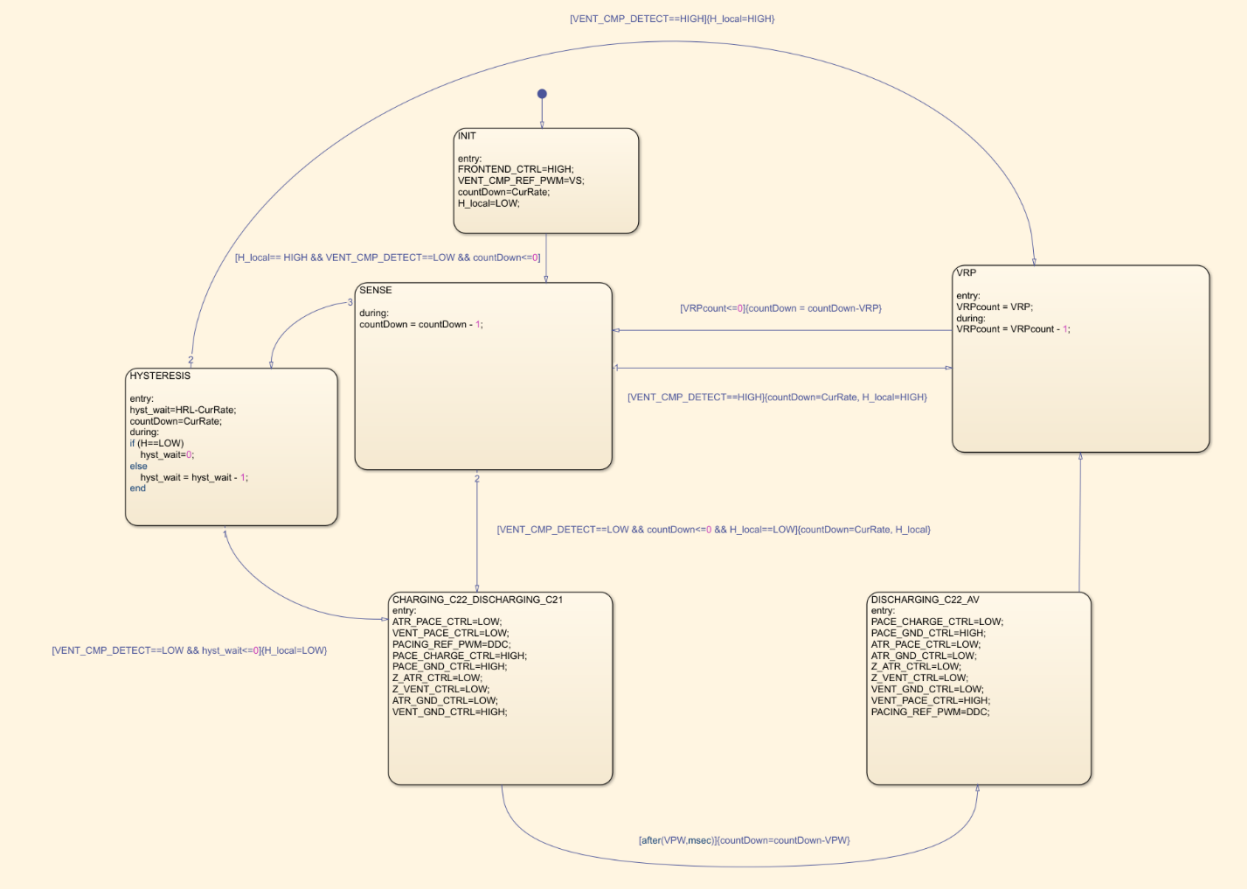


Figure 8: Simulink Model of VVI.

Table 21: VVI Simulink Model Behaviour for Case 1.

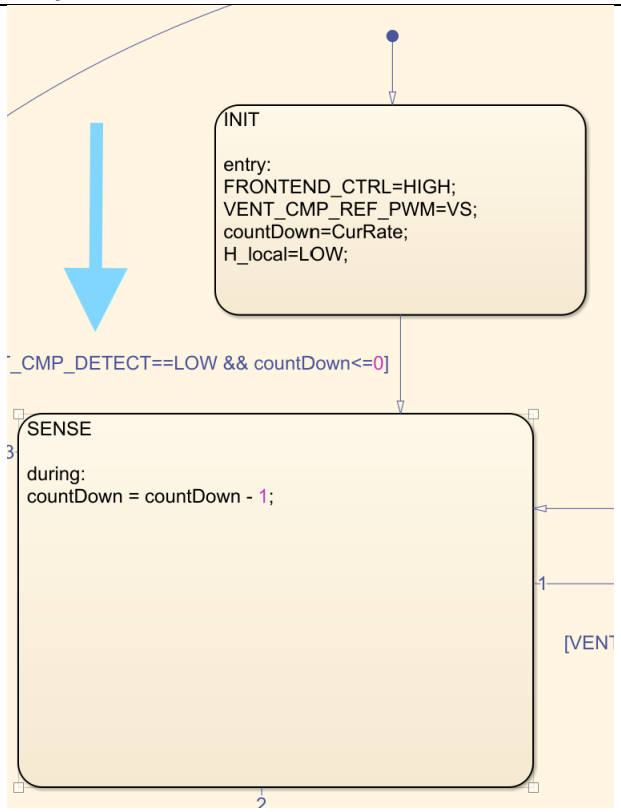
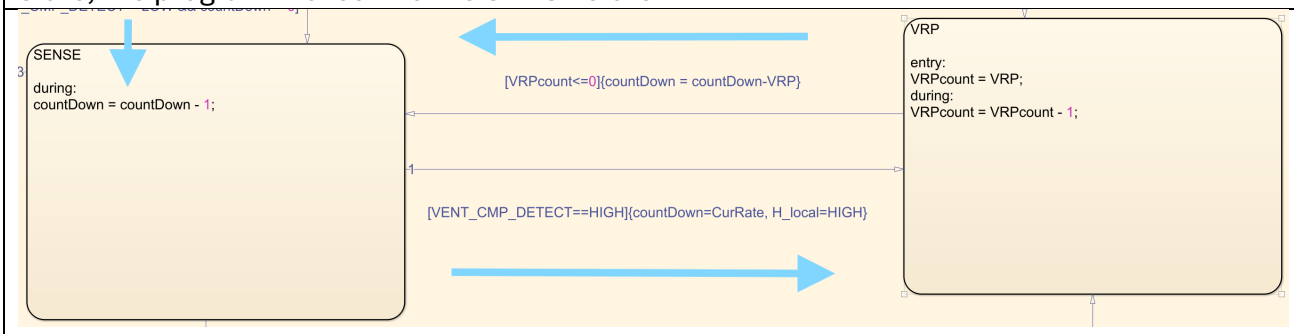
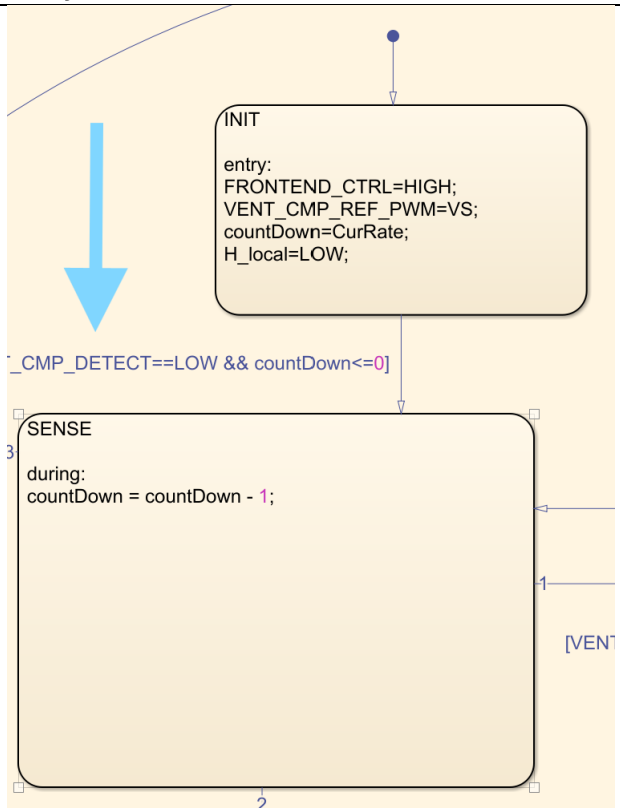
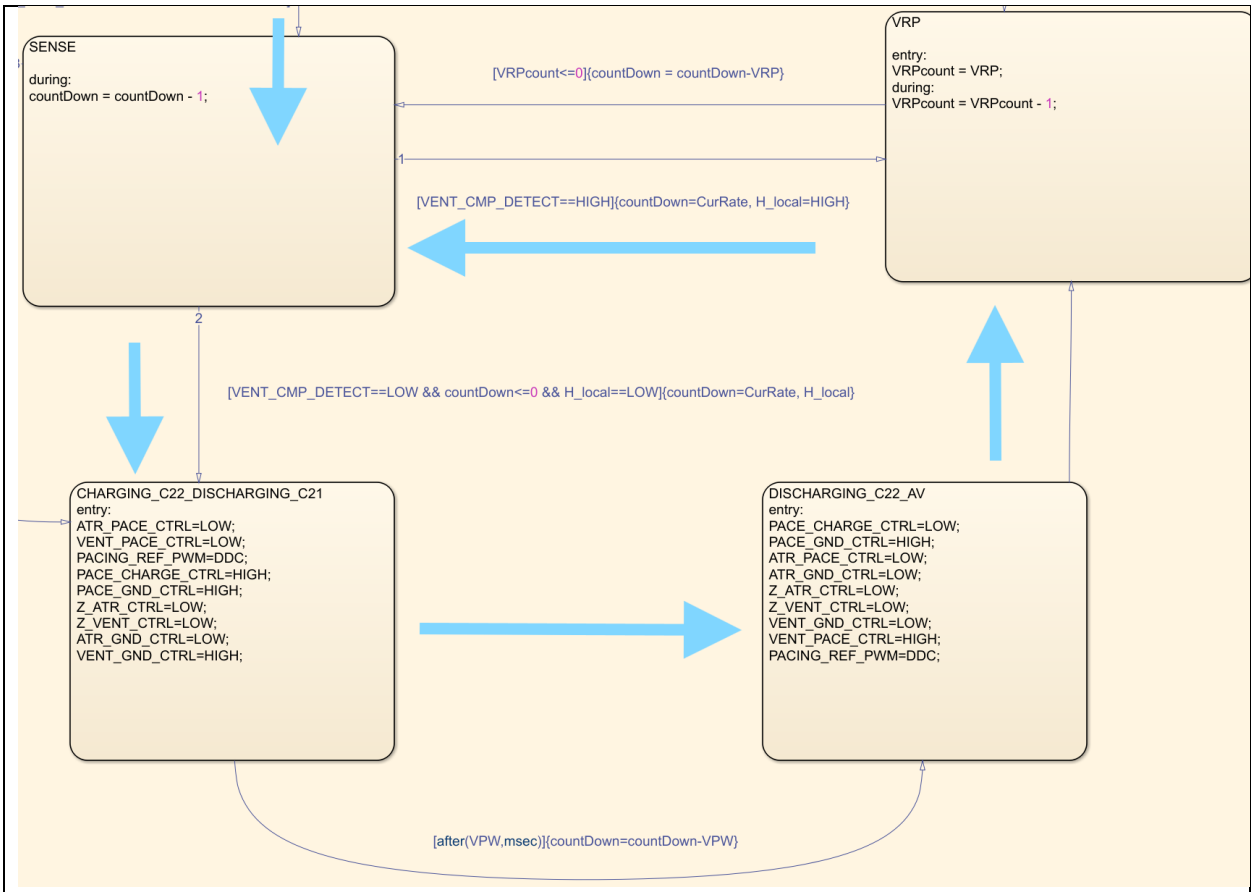
Case 1: hysteresis OFF, natural heart rate >= current rate
 The entry point sets the necessary pins for pacing/sensing in the INIT state. After the INIT state, the program moves into the SENSE state.
 Within the SENSE state, a countdown begins from the current rate representing the amount of time for a single heartbeat of the current rate. If a natural heartbeat is detected before the next pace (if countdown is not 0 when a heartbeat is detected), the program moves into the VRP state. Within this state, no sensing occurs for the duration of the ventricular refractory period. Upon exiting the refractory period, the countdown timer is set to the amount of time left over after the refractory period, and sensing occurs. This cycle will repeat as long as the natural heartbeat of the ventricle is greater than the current rate.

Table 22: VVI Simulink Model Behaviour for Case 2

Case 2: hysteresis OFF, natural heart rate < current rate
 The entry point sets the necessary pins for pacing/sensing in the INIT state. After the INIT state, the program moves into the SENSE state.



Within the SENSE state, a countdown begins from the current rate representing the amount of time for a single heartbeat of the current rate. If no natural heartbeat is detected during the period a heartbeat of the current rate (when countdown reaches 0), an artificial pulse is initiated. The countdown is reset to the total duration of a heartbeat in the current rate.

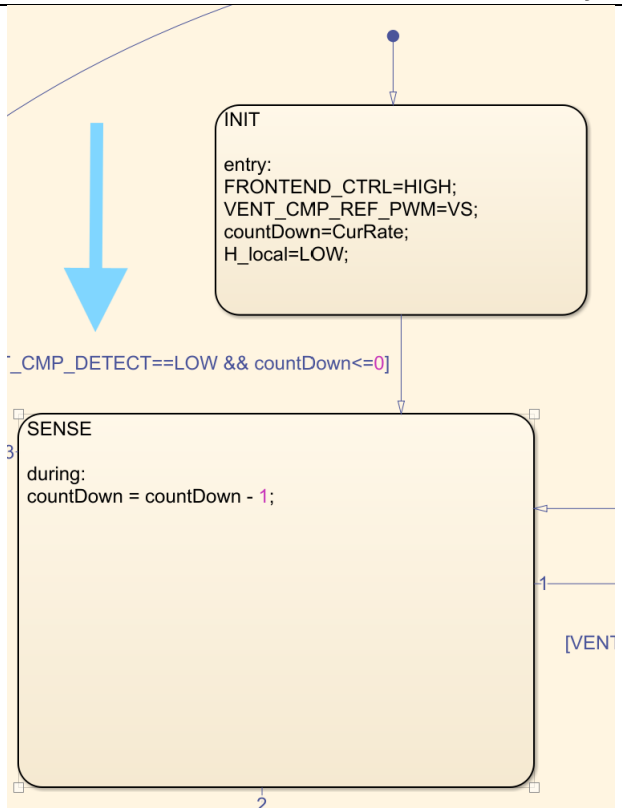
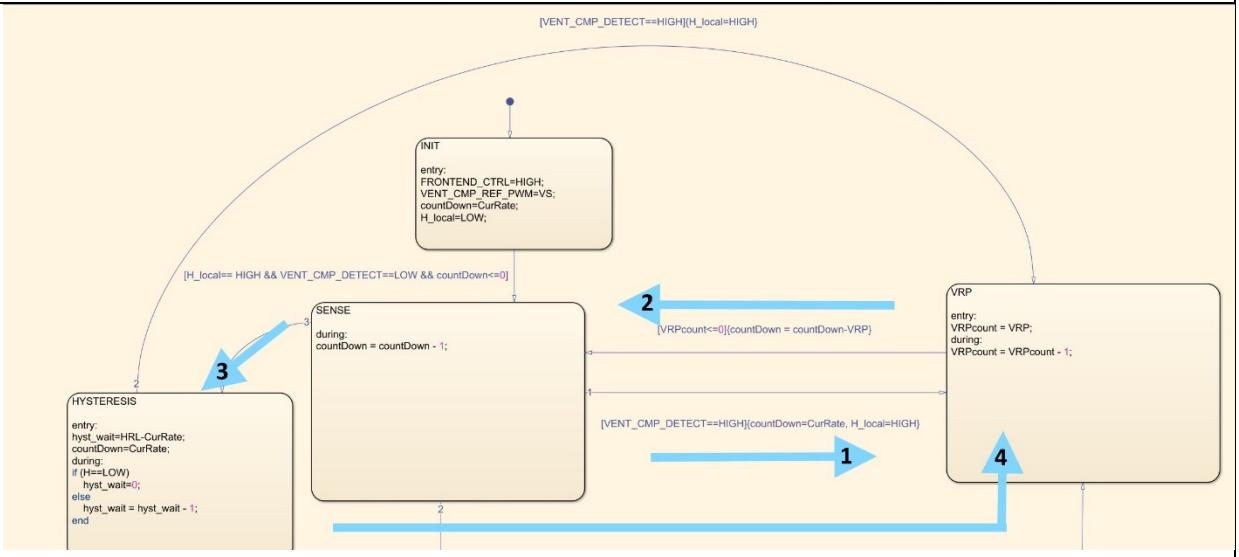
The program moves into the CHARGING_C22_DISCHARGING_C21 state, where capacitors C22 is charging and C21 is discharging. This sequence allows for the pacemaker to prepare to administer a pulse to the ventricle. This sequence lasts for the ventricular pulse width period, to allow the capacitor to charge to provide the correct pulse width.

The program moves into the DISCHARGING_C22_AV state to administer the pulse to the ventricle by discharging C22.

After administering the pulse, the program moves to the VRP state where no sensing occurs for the duration of the ventricular refractory period. The countdown timer is set to wait in the SENSE state for the remainder of period of a heartbeat of the current rate.

This cycle repeats if no heartbeat is detected during the duration of the SENSE state, when the natural heart rate is lower than the current rate of the pacemaker.

Table 23: VVI Simulink Model Behaviour for Case 3

Case 3: hysteresis ON, current rate > natural heart rate > hysteresis rate limit
 The entry point sets the necessary pins for pacing/sensing in the INIT state. After the INIT state, the program moves into the SENSE state.
 Within the SENSE state, a countdown begins from the current rate representing the amount of time for a single heartbeat of the current rate. If a heartbeat is detected before the next artificial pulse, the program moves into the VRP state. During this transition,

countdown is reset to the duration of a single heartbeat and hysteresis is enabled via H_local.

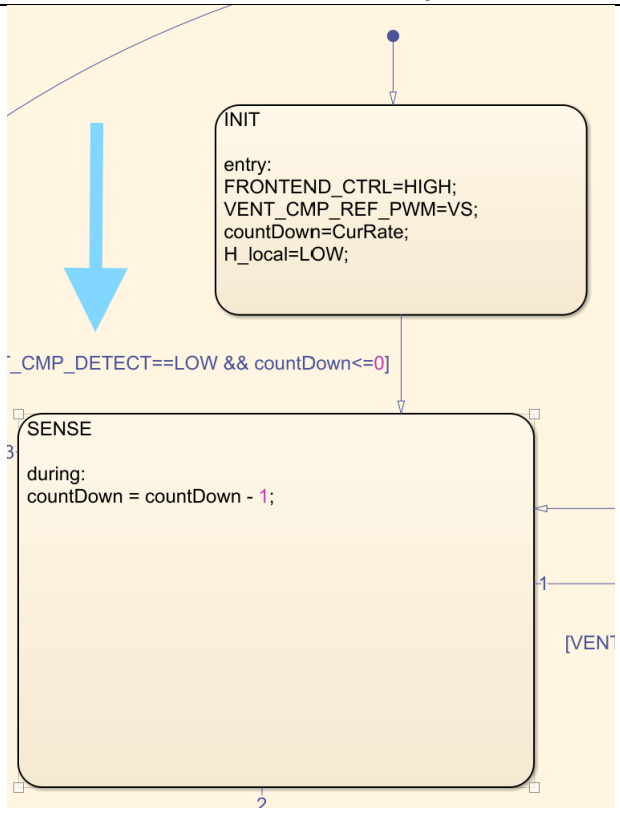
Within the VRP state, no sensing occurs for the duration of the ventricular refractory period. Upon exiting the refractory period, the countdown timer is reduced to the amount of time leftover after the refractory period, and sensing occurs.

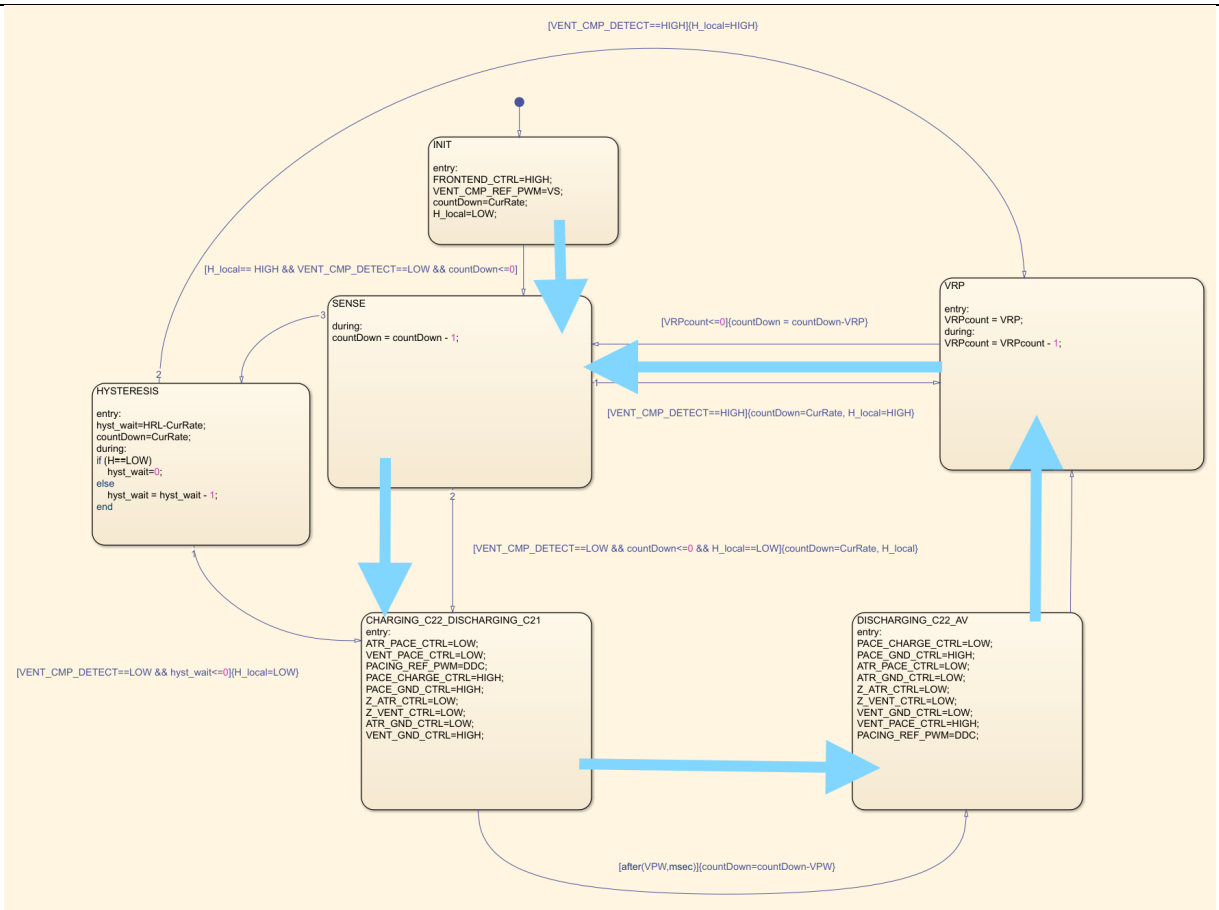
If during the sensing state, a heartbeat isn't detected (if countdown goes to 0), then the program moves into the HYSTERESIS state.

Within the HYSTERESIS, the pacemaker senses for a heartbeat for an additional time interval. This interval is defined by the difference between the hysteresis rate limit and the current rate.

If a heartbeat is detected during this period, the program will go to the VRP state. This entire cycle continues as long as the natural heart rate is in between the hysteresis rate limit and the current rate.

Table 24: VVI Simulink Model Behaviour for Case 4

Case 4: Hysteresis on, natural heart rate < hysteresis rate < current rate
 The entry point sets the necessary pins for pacing/sensing in the INIT state. After the INIT state, the program moves into the SENSE state.



Within the SENSE state, a countdown begins from the current rate representing the amount of time for a single heartbeat of the current rate. If no natural heartbeat is detected during the period a heartbeat of the current rate (when countdown reaches 0), the pacemaker administers an artificial pulse, beginning with the CHARGE_C22_DISCHARGING_C21 state. The local flag H_local is still LOW, meaning that the HYSTERESIS state is skipped.

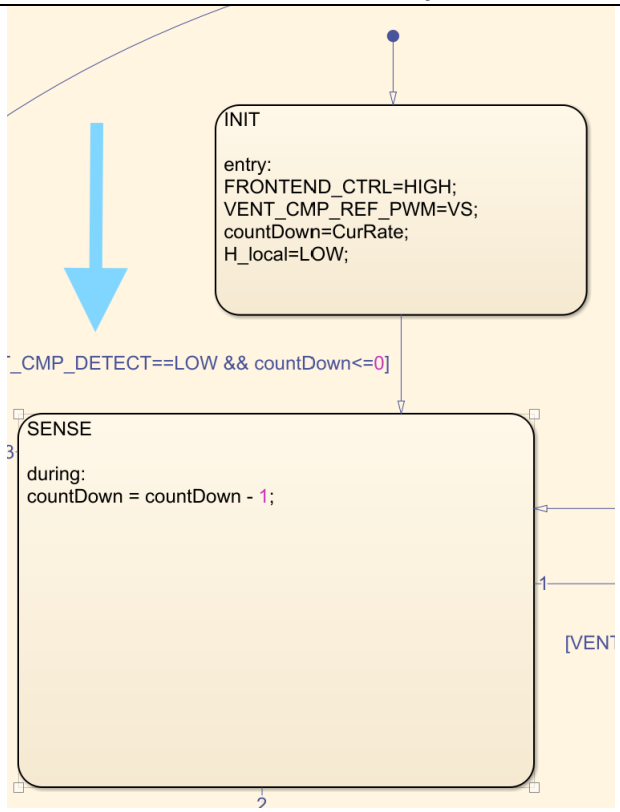
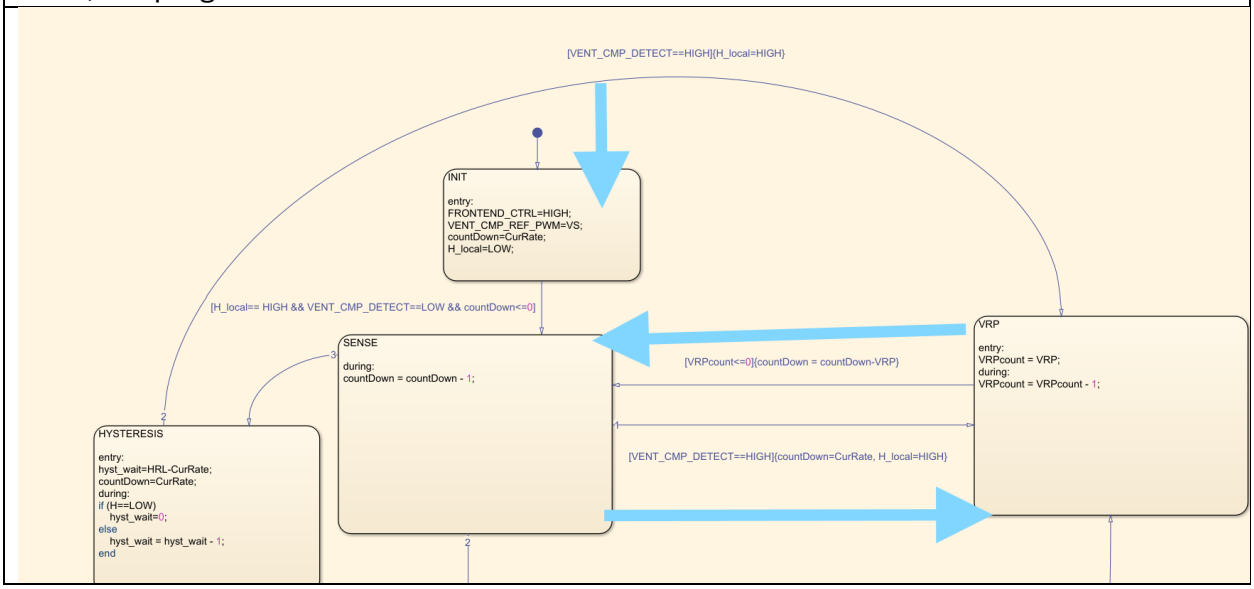
In the CHARGE_C22_DISCHARGING_C21 state, capacitors C22 is charging and C21 is discharging. This sequence allows for the pacemaker to prepare to administer a pulse to the ventricle. This sequence lasts for the ventricular pulse width period, to allow the capacitor to charge to provide the correct pulse width.

The program moves into the DISCHARGING_C22_AV state to administer the pulse to the ventricle by discharging C22.

After administering the pulse, the program moves to the VRP state where no sensing occurs for the duration of the ventricular refractory period. The countdown timer is set to wait in the SENSE state for the remainder of period of a heartbeat of the current rate.

This cycle repeats if no heartbeat is detected during the duration of the SENSE state, when the natural heart rate is much lower than the hysteresis and current rate of the pacemaker.

Table 25: VVI Simulink Model Behaviour for Case 5

Case 5: Hysteresis on, natural heart rate > hysteresis rate > current rate

The entry point sets the necessary pins for pacing/sensing in the INIT state. After the INIT state, the program moves into the SENSE state.


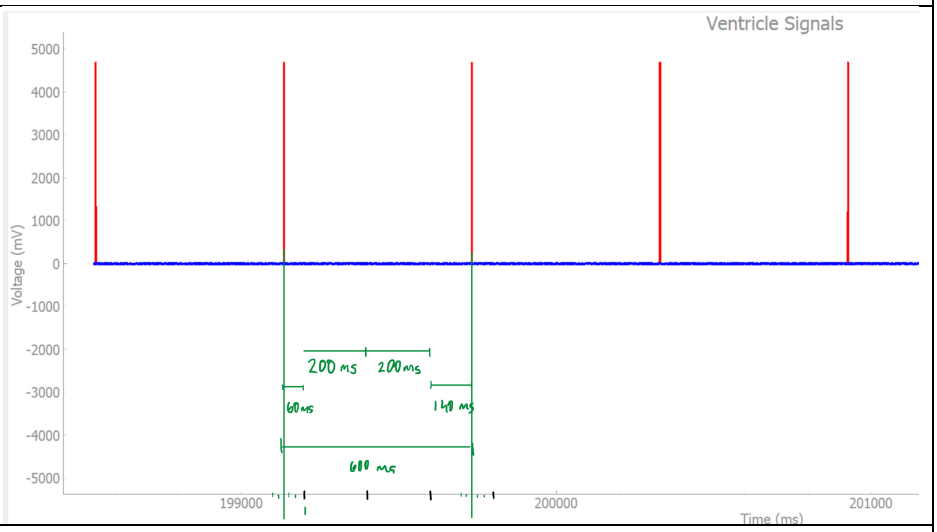
Within the SENSE state, a countdown begins from the current rate representing the amount of time for a single heartbeat of the current rate. If a natural heartbeat is detected before the next pace (if countdown is not 0 when a heartbeat is detected), the program moves into the VRP state. The flag H_local is set to HIGH indicating that a spontaneous heartbeat occurred.

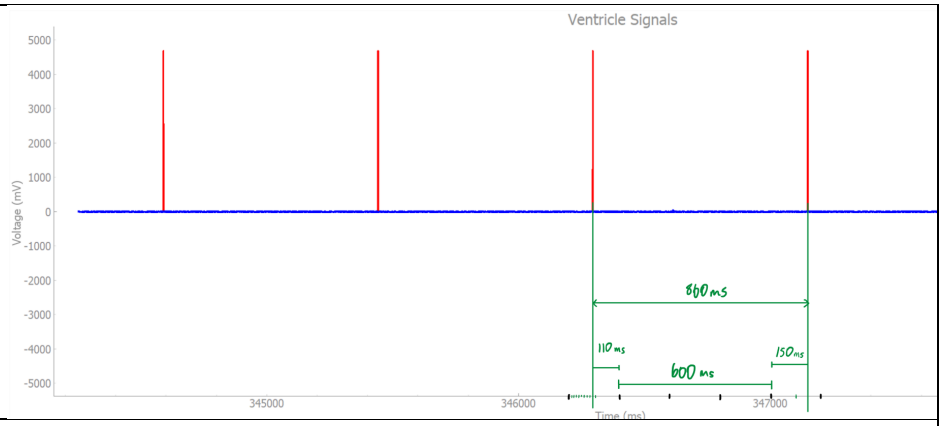
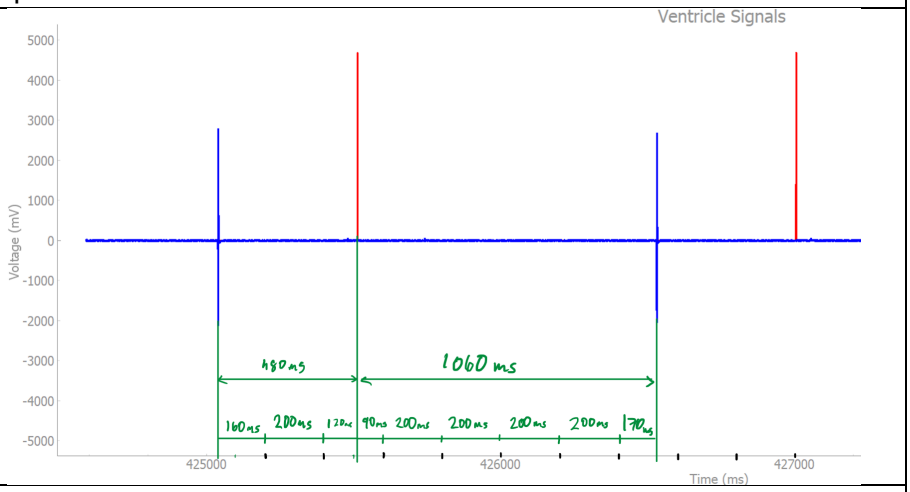
Within this state, no sensing occurs for the duration of the ventricular refractory period. Upon exiting the refractory period, the countdown timer is set to the amount of time leftover after the refractory period, and sensing occurs. This cycle will repeat as long as the natural heartbeat of the ventricle is greater than the current rate, meaning that hysteresis even though enabled, won't occur.

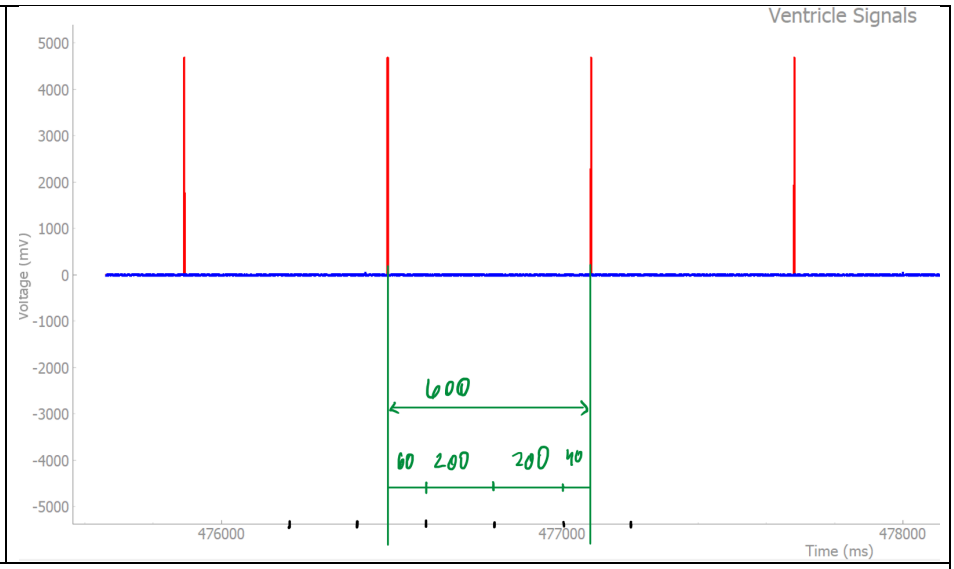
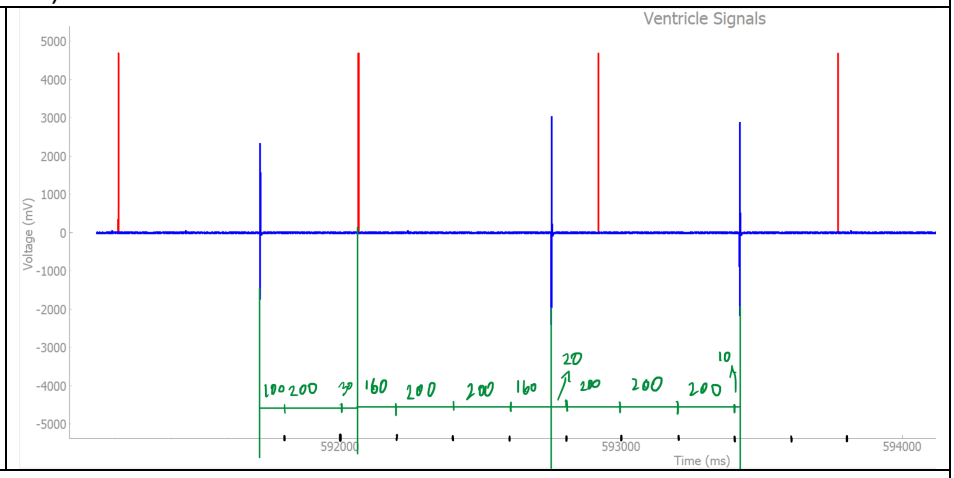
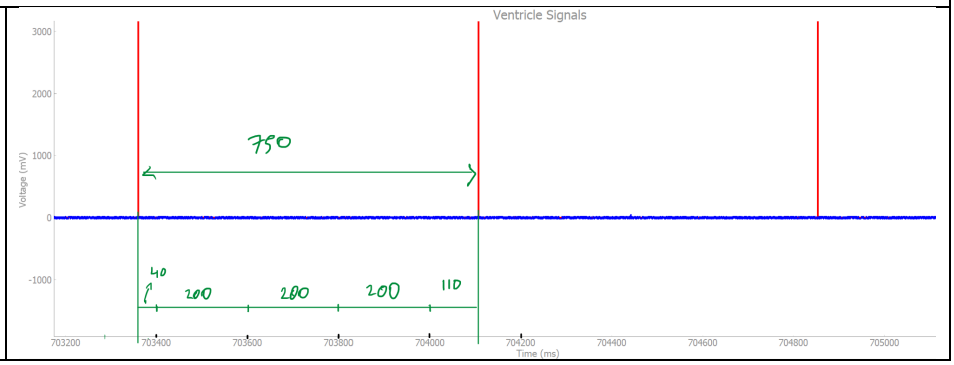
5.3.5 Heart View Tests

The mode was tested with heart view at various parameters to ensure functionality.

Table 26: Heart View results for VVI

Test Case	Result
CurrRate = 90 ppm VPW = 1 ms VRP = 320 ms Hysteresis ON HRL = 60 ppm Natural Heart Rate = 100 ppm	
As the natural heart rate is greater than the current rate, when hysteresis is on, the pacemaker should always inhibit the pace, allowing the heart to beat at 100 ppm (600 ms per heartbeat).	

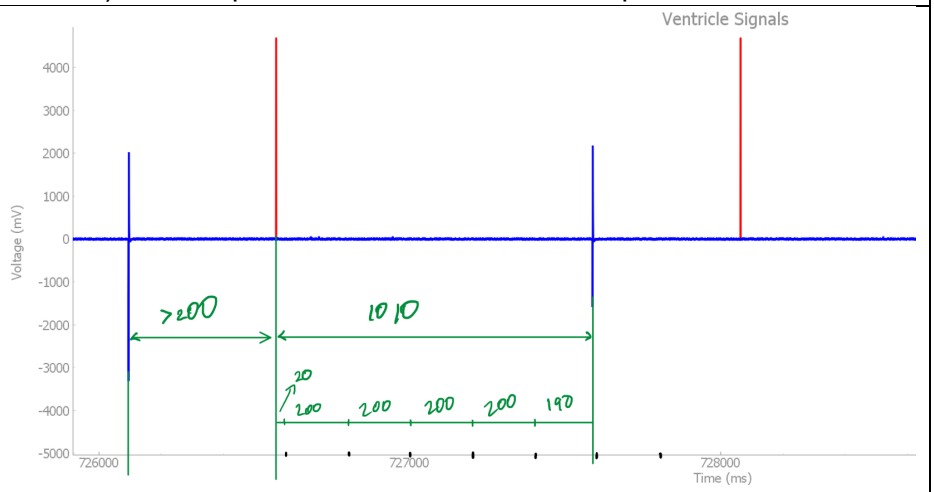
CurrRate = 90 ppm VPW = 1 ms VRP = 320 ms Hysteresis ON HRL = 60 ppm Natural Heart Rate = 70 ppm	
As the natural heartbeat is 70 ppm which is in between the current rate and the hysteresis rate limit, the pacemaker provides an additional period of sensing. As a result, there is no pacing by the pacemaker.	
CurrRate = 90 ppm VPW = 1 ms VRP = 320 ms Hysteresis ON HRL = 60 ppm Natural Heart Rate = 40 ppm	
As the natural heartbeat is 40 ppm which is below the hysteresis rate limit, the pacemaker will pace at a rate of 90 BP ppm (1500 ms). It is seen that a heartbeat is sensed after the refractory period (320 ms). Therefore, it is expected for the pacemaker to inhibit the pulse and wait and additional time interval before it can pace, if there is no spontaneous heartbeat.	

CurrRate = 90 ppm VPW = 1 ms VRP = 320 ms Hysteresis OFF HRL = 60 ppm Natural Heart Rate = 100 ppm	
As the natural heartrate is 100 ppm, which is higher than the current rate with hysteresis disabled, the pacemaker will not pace and the heart should maintain its heartrate (100 ppm = 600ms/heartbeat).	
CurrRate = 90 ppm VPW = 1 ms VRP = 320 ms Hysteresis OFF HRL = 60 ppm Natural Heart Rate = 70 ppm	
As the natural heartrate is lower than the current rate, the pacemaker is expected to pace the heart at the rate of 90 ppm. It is seen that if a natural heartbeat is sensed outside the refractory period, the pacemaker will inhibit the pace and wait for a duration of a heartbeat (90 ppm = 667ms) before administering a pulse, provided no spontaneous heartbeat is detected.	
CurrRate = 60 ppm VPW = 0.4 ms VRP = 200 ms Hysteresis ON HRL = 60 ppm Natural Heart Rate = 80 ppm	

As the natural heartrate is greater than the current rate, the heart will continue to beat at 80 ppm (750ms per heartbeat) and the pacemaker will not deliver a pulse.

CurrRate = 60 ppm
VPW = 0.4 ms
VRP = 200 ms
Hysteresis ON
HRL = 60 ppm

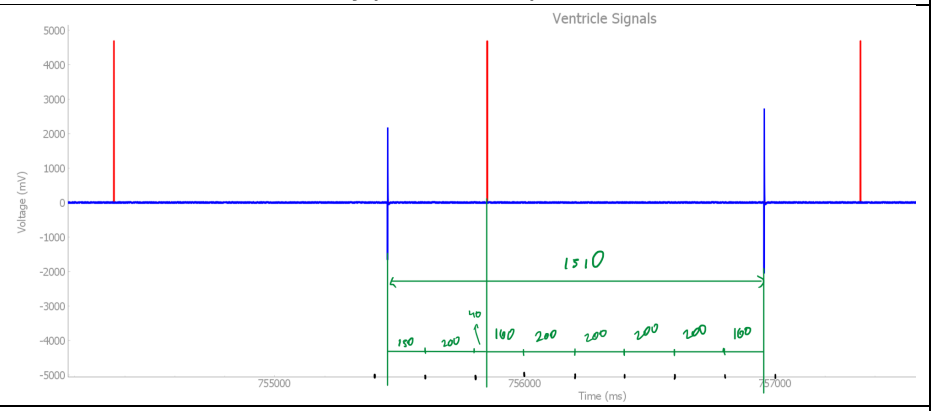
Natural Heart Rate = 40 ppm



As the natural heartrate is less than the current rate, the pacemaker will pulse at 60 ppm. When a heartbeat is detected after the refractory period, the pace is inhibited.

CurrRate = 60 ppm
VPW = 0.4ms
VRP = 500 ms
Hysteresis ON
HRL = 60 ppm

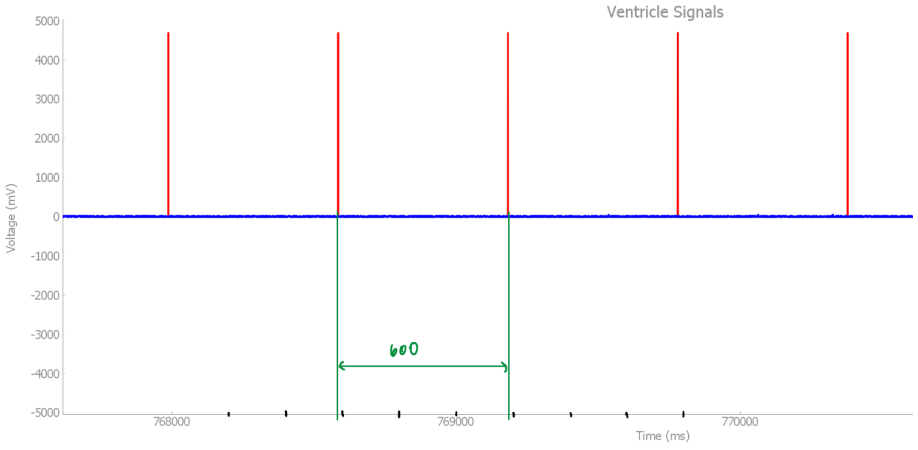
Natural Heart Rate = 40 ppm



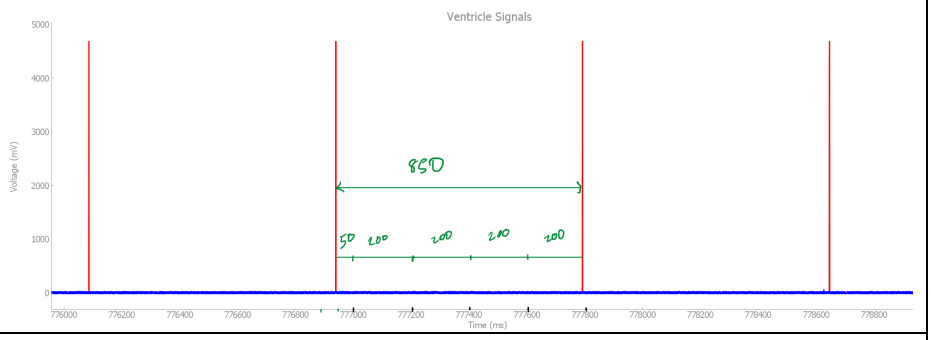
As the natural heartrate is less than the current rate, the pacemaker will pulse at 60 ppm. The pacemaker doesn't inhibit the pulse this time as the refractory period is larger.

CurrRate = 60 ppm
VPW = 0.4 ms
VRP = 500 ms
Hysteresis ON
HRL = 60 ppm

Natural Heart Rate = 100 ppm



As the natural heartrate is greater than the current rate, the pacemaker will not pace, and the heart will maintain its rate.

CurrRate = 60 ppm VPW = 0.4 ms VRP = 500 ms Hysteresis OFF HRL = 60 ppm Natural Heart Rate = 70 ppm	
As the natural heartrate is greater than the current rate, the pacemaker will not pace, and the heart will maintain its rate.	

5.3.6 Validation and Verification

This testing confirms that the VVI mode operates independently of the ventricle's intrinsic activity and can function at higher pacing rates. Additionally, the VVI mode adheres the hysteresis rate limit when waiting the additional time interval after the SENSE state. However, inconsistencies were observed between duration of some artificial pulses by several milliseconds especially when the natural heart rate is considerably lower than the current rate and hysteresis rate limit. These discrepancies may result from the non-ideal behavior of the equipment. Further testing is required to pinpoint the exact cause of these variations.

5.3.7 Design Justification

VVI offers hysteresis functionality. However, the user has the option to toggle it on or off. Therefore, the mode should remain functional without hysteresis. Therefore, the 'SENSE' state is independent of hysteresis. The 'HYSTERESIS' state includes added functionality which allows prolonged sensing upon a sensed natural ventricular event.

For pacing the ventricle, charging the capacitor prior to pulse delivery ensures that sufficient energy is available to effectively pace the heart when needed. Discharging the pulse into another capacitor allows the pacemaker to safely manage and ground the excess charge. By setting the interval between pulse preparation and delivery to match the difference between the pacing rate and the ventricular pulse width, the pacemaker optimizes timing, avoiding unnecessary delays. Additionally, waiting the ventricular refractory period after delivering the pulse mirrors atrium's natural activity, ensuring efficient and reliable pacing.

5.3.8 Requirement Changes

1. Must take a user set ventricular pulse amplitude between 3.5–7.0 V.

5.3.9 Design Decision Changes

1. Implement safety checks for all input variables so that programmable parameters are limited to certain allowable values.
2. Ensure proper time interval after “SENSE” state when hysteresis is on.
3. Ensure proper duration between artificial pulses and natural heartbeats.

5.4 AAI

5.4.1 Description

In AAI mode, the pacemaker device paces and senses the atrium. Detection of a heartbeat above a specified threshold inhibits artificial pacing. A state flow diagram was developed in Simulink for this module. The following parameters are variables are input into this module

5.4.2 Requirements

7. Must take a user set heart rate to pulse at
8. Must take a user set pulse width, amplitude, lower rate limit, upper rate limit, hysteresis mode, sensitivity, refractory period, rate smoothing condition
9. Must not pulse the atrium when it beats naturally above the current rate
10. Must pulse the atrium when it beats below the current rate
11. Must not sense during the atrial refractory period
12. Must allow for hysteresis upon natural atrial events when hysteresis is active

5.4.3 Variables and Constants

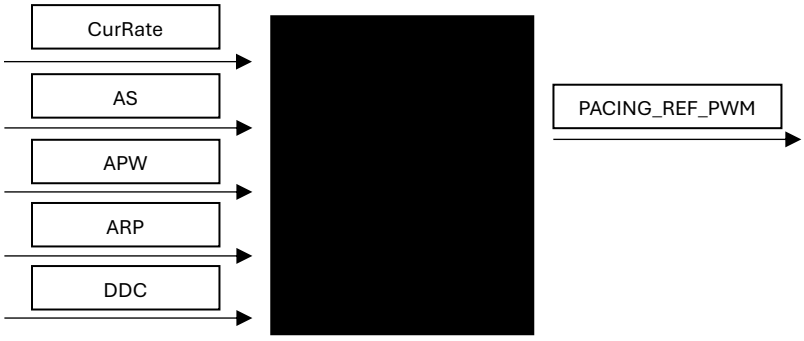


Figure 9: Black Box Diagram for AAI

5.4.3.1 Monitored Variables

Table 27: Monitored Variables for AAI

Variable	Name	Range	Nominal Value	Description
Current Rate	CurRate	LRL-URL ppm	-	Current ppm which pulses are being delivered
Atrial Pulse Width	APW	0.1-1.9 ms	0.4 ms	Width of the pulse delivered to the atrium
Desired Duty Cycle	DDC	0-100 %	$\mathrm{DDC} = \frac{\mathrm{AA}}{\mathrm{Max\ Volts}} * 100\%$	Percentage of time a signal is HIGH in a PWM system

			$\mathrm{DDC} = 70\%$	

5.4.3.2 Control Variables

Table 28: Controlled Variables for AOO

Variable	Name	Range	Nominal Value	Description
Pacing Reference Pulse Width Modulation	PACING_REF_PWM	0-100 %	DDC	Percentage of time a signal is HIGH in a PWM system to charge primary capacitor

5.4.3.3 Programmable Parameters

In AAI mode, the pacemaker device paces and senses the atrium. Detection of a heartbeat above a specified threshold inhibits artificial pacing. A state flow diagram was developed in Simulink for this module. The following parameters are variables are input into this module

Table 29: Programmable Parameters for AAI

Variable	Name	Range	Nominal Value	Description
Lower Rate Limit	LRL	50-90 ppm	60 ppm	Minimum ppm at which pulses are delivered in the absence of sensed activity
Upper Rate Limit	URL	50-175 ppm	120 ppm	Maximum ppm at which pulses are delivered in the absence of sense activity
Current Rate	CurRate	LRL-URL ppm	-	Current ppm which pulses are being delivered
Atrial Amplitude	AA	3.5–7.0 V	3.5 V	***
Atrial Pulse Width	APW	0.1-1.9 ms	0.4 ms	Width of the pulse delivered to the atrium
Atrial Sensitivity	AS	1.0-10 mV**	0.5 mV	Voltage at which to compare the sensed pulse to allow signal to pass

Atrial Refractory Period	ARP	150-500 ms	250 ms	Period following a natural or artificial pulse where sensing is disabled
Post Ventricular Atrial Refractory Period	PVARP	150-500 ms	250 ms	Period after a artificial or natural-sensed ventricular event during which atrial sensing is disabled
Hysteresis	H	HIGH or LOW	-	HIGH if hysteresis pacing is enabled. LOW if hysteresis pacing is disabled
Hysteresis Rate Limit	HRL	***	-	Lowest ppm at which hysteresis may inhibit artificial pacing
Rate Smoothing	RS	Off, 3, 6, 9, 12, 15, 18, 21	Off	Maximum allowable change in heartrate in percentage
Desired Duty Cycle	DDC	0-100 %	$\mathrm{DDC} = \frac{\mathrm{AA}}{\mathrm{Max\ Volts}} * 100\%$ $\mathrm{DDC} = \frac{3.5\ \mathrm{V}}{5\ \mathrm{V}} * 100\%$ $\mathrm{DDC} = 70\%$	Percentage of time a signal is HIGH in a Pulse Width Modulation (PWM) system

During AAI, the pacemaker will actively sense the heart for intrinsic activity. If the atrium is sensed to be beating above the Current Rate (CurRate), artificial pacing will be inhibited. Otherwise, if the atrium is beating below the Current Rate, the heart will pace to align the rate with the Current Rate.

Hysteresis Pacing has also been implemented into this mode. Hysteresis pacing extends the sensing period following a natural pacing event from the atrium. When activated, it senses the LRL to allow the heart to beat naturally. If no further natural activity is detected, it continues pacing at the Current Rate.

5.4.4 Simulink Explanation

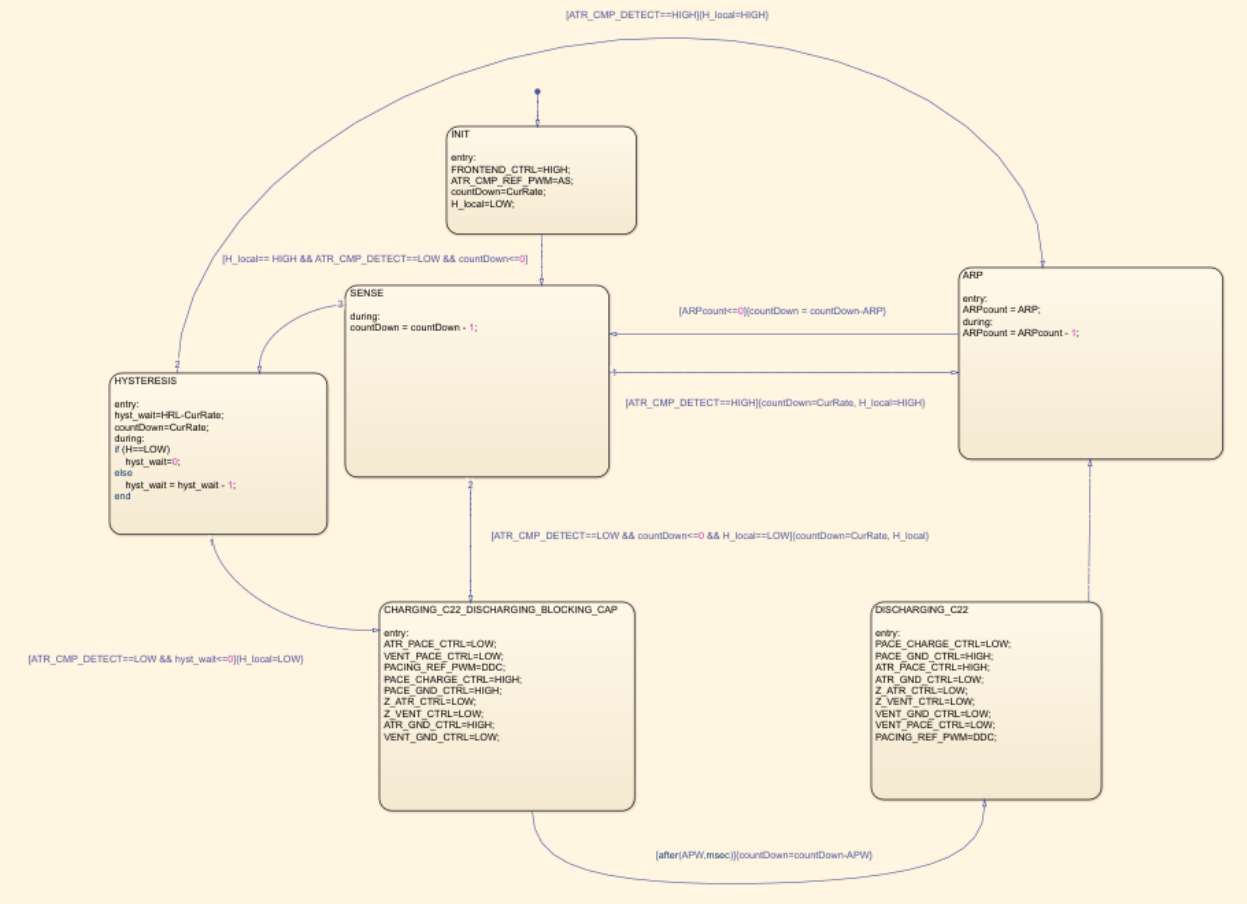
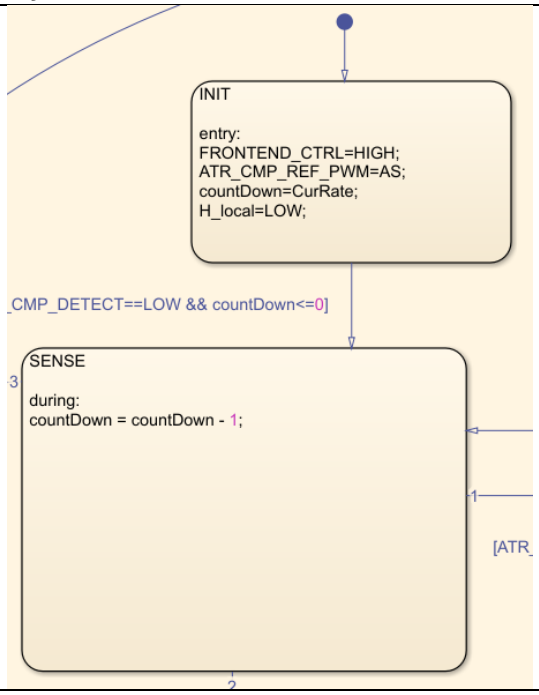
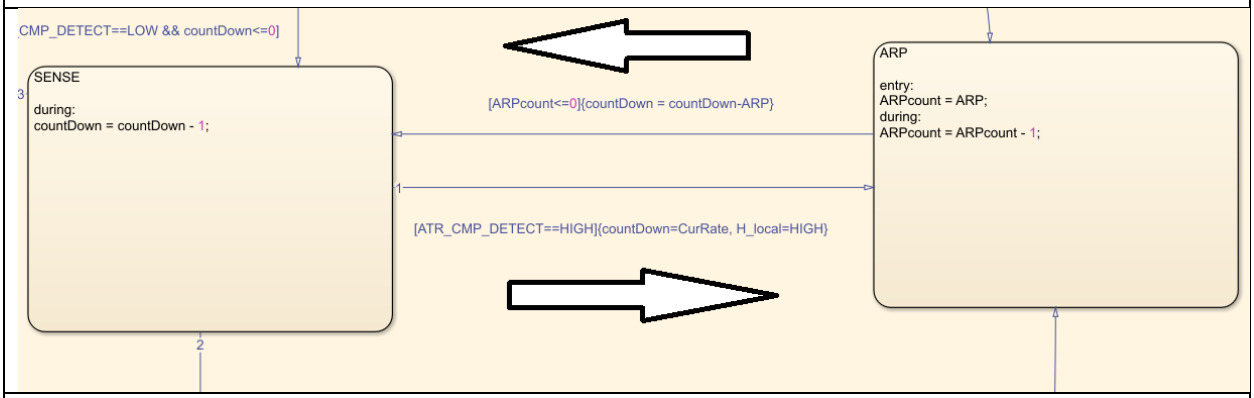
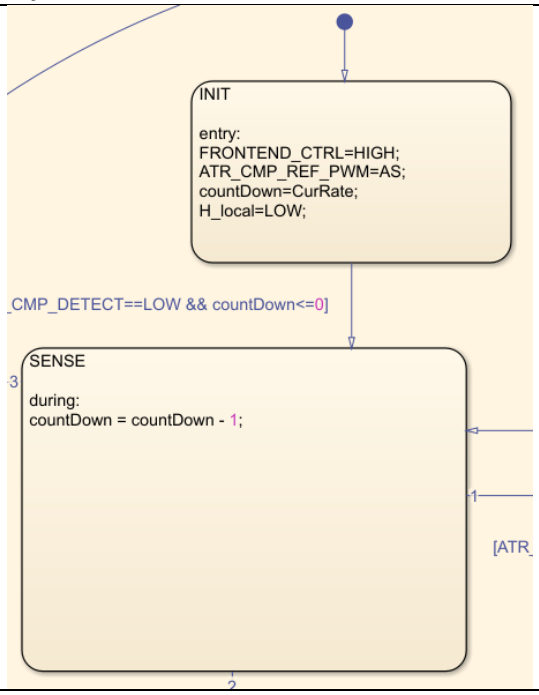


Figure 10: Simulink Model of AAI

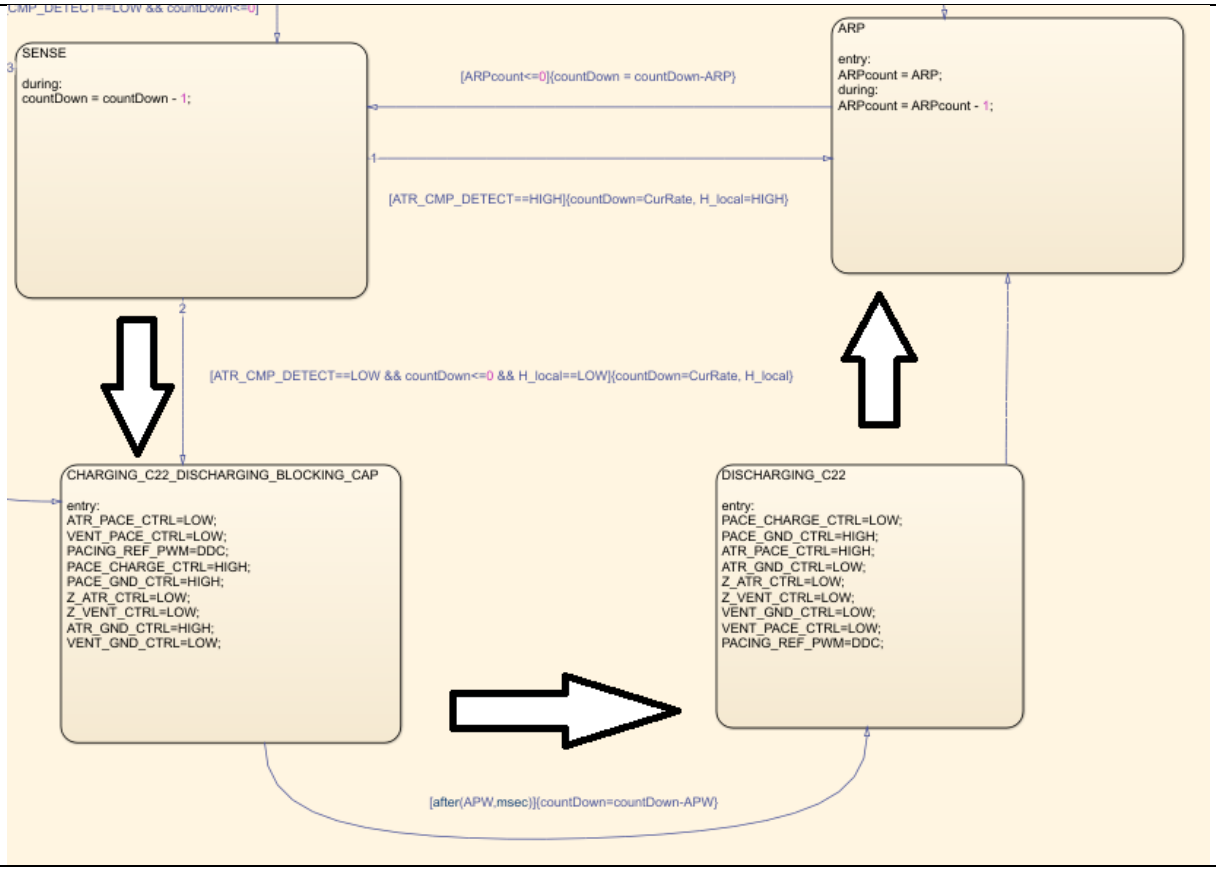
Table 30: AAI Simulink Model Behaviour

Case 1: hysteresis off, natural heart rate > current rate

The entry point sets the necessary pins for pacing/sensing. The 'SENSE' block is then entered which begins a countdown from the Current Rate in msec/beat. A pulse is sense before 'countDown' reaches zero.

Upon sensing a natural pulse from the atrium, the atrial refractory period (ARP) is entered during which no sensing occurs. Upon exiting the refractor period, the countdown timer is reset, and sensing occurs. This cycle will repeat as long as the atrium pulses at a rate greater than the current rate.

Case 2: hysteresis off, natural heart rate < current rate

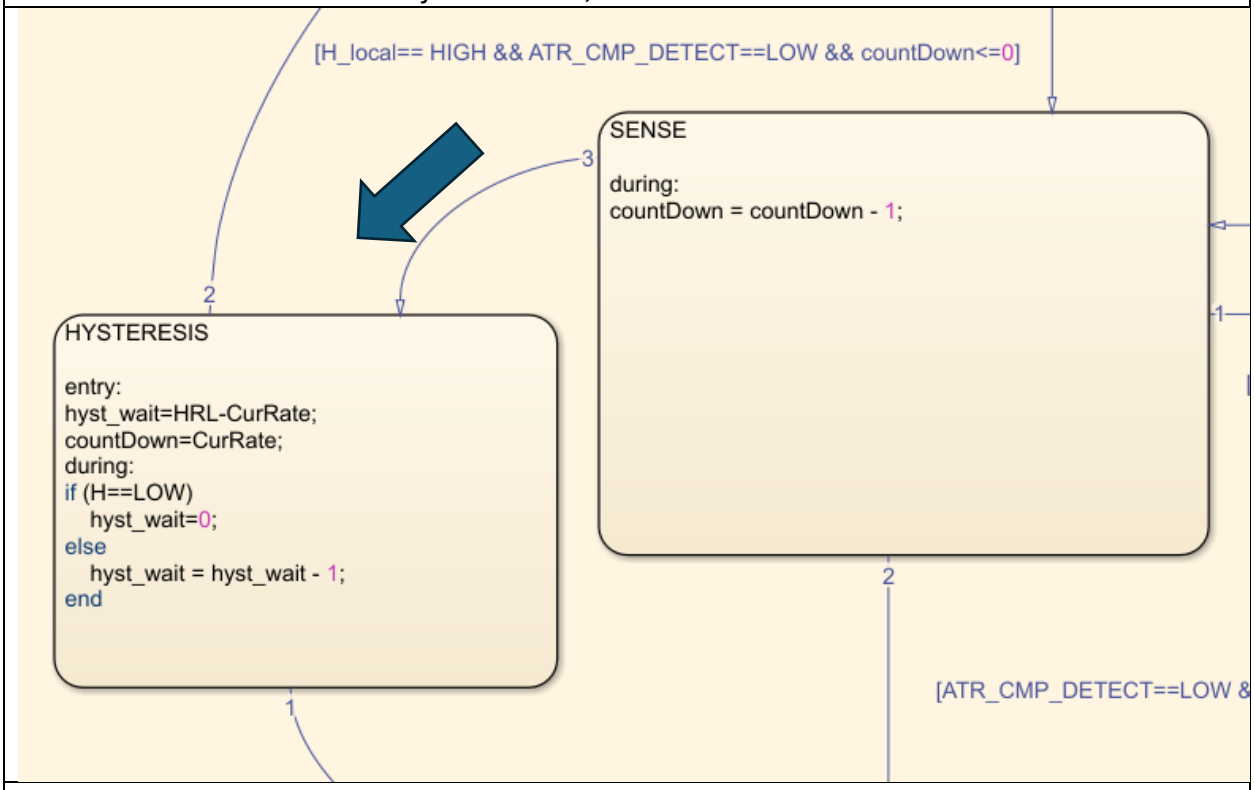


The mode begins sensing upon entry. Countdown is set to the msec corresponding to the duration of a heartbeat set to 'Cur Rate'.



If an atrial event is not sensed before ‘countDown’ reaches 0, a pulse will be sent to the heart. Upon entry into the pacing state, the countdown timer will reset. The pacing state leads to the atrial refractory period which eventually leads back to the sensing period. As long as no pulse is sensed. The pacemaker will send pulses to the atrium at a rate corresponding to ‘CurRate’.

Case 3: hysteresis on, natural atrial event sensed

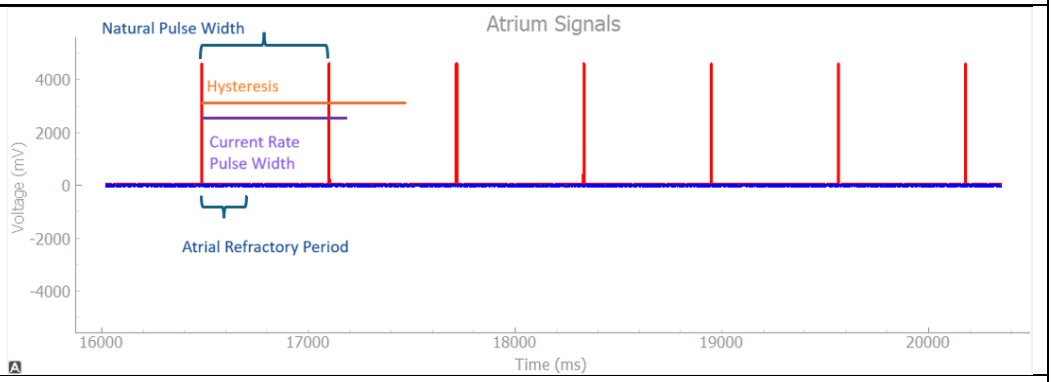
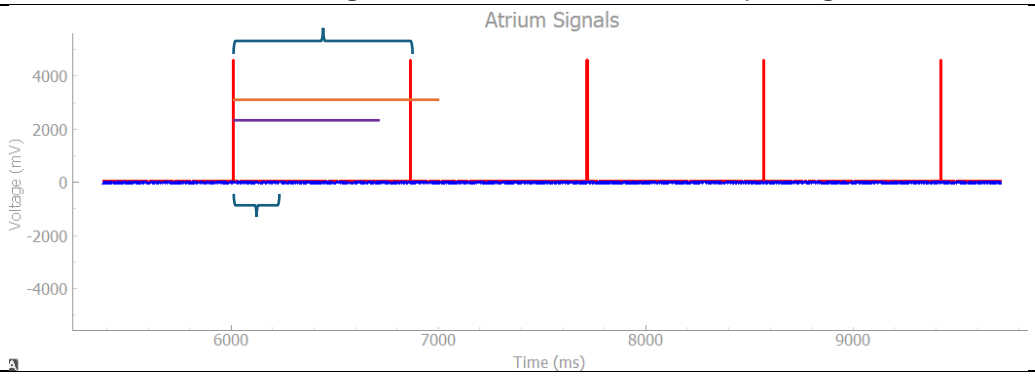
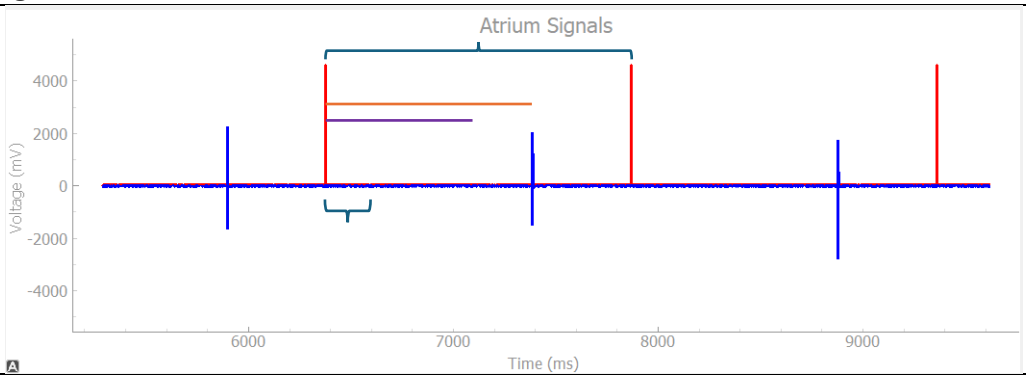


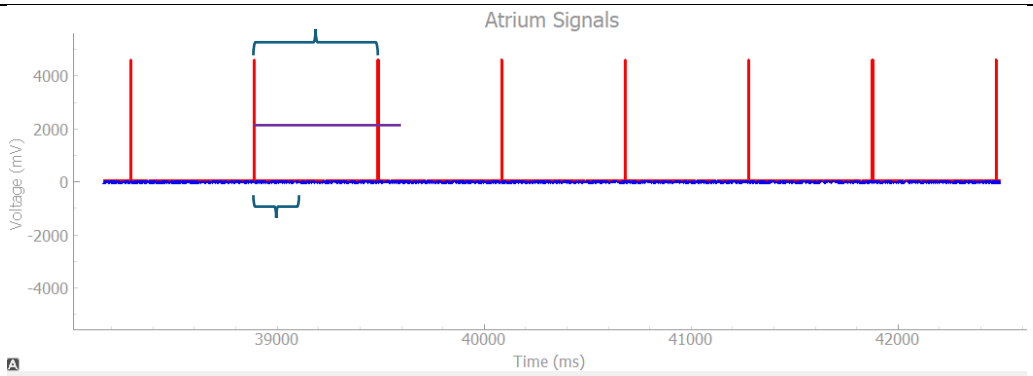
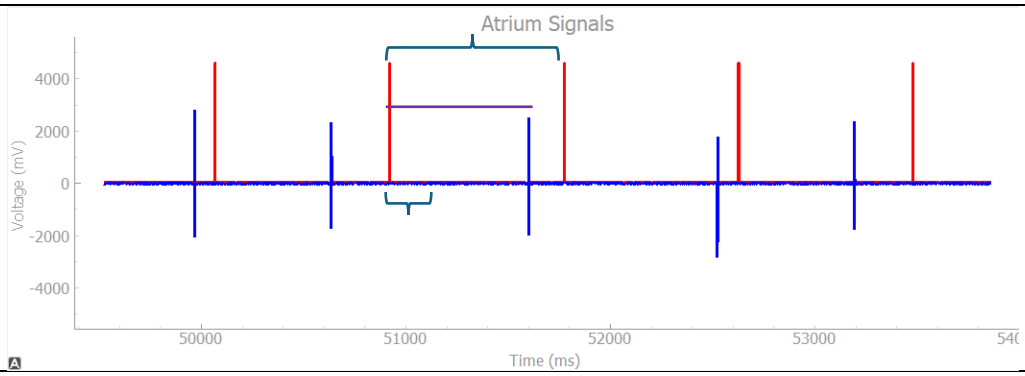
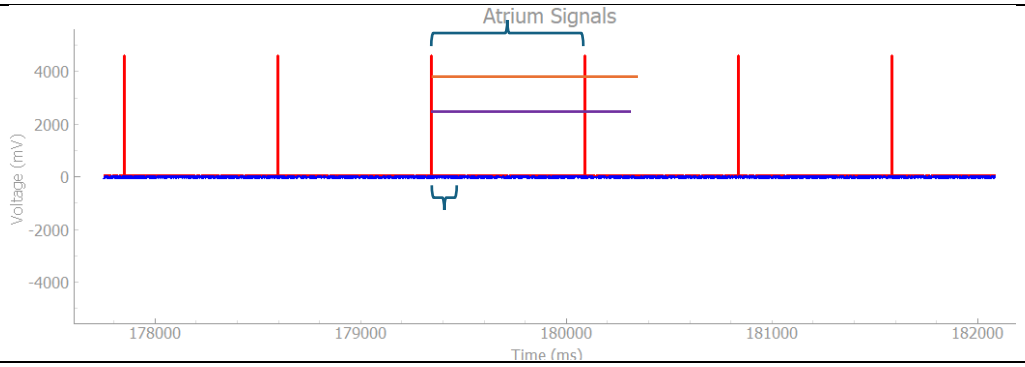
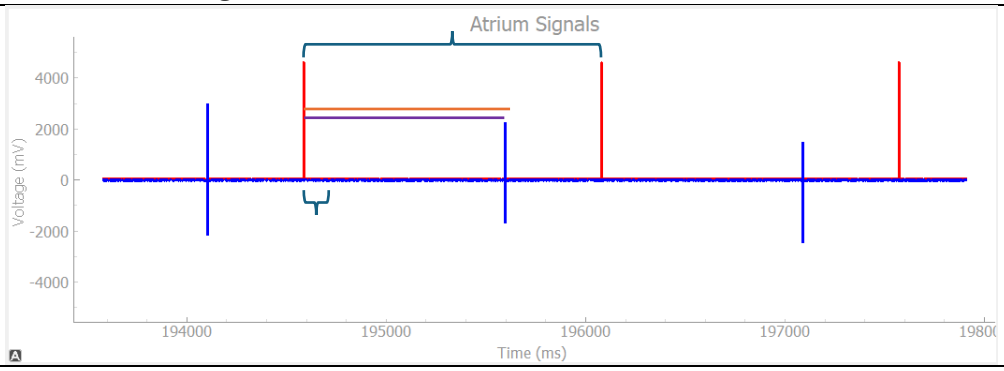
When hysteresis is on and a natural atrial event Is sensed, ‘H_local’ is set to HIGH. This causes the ‘SENSE’ state to transition into the ‘HYSTERESIS’ state. The ‘HYSTERESIS’ state is identical in functionality to the ‘SENSE’ state except that it now senses at a period corresponding to the ‘LRL’.

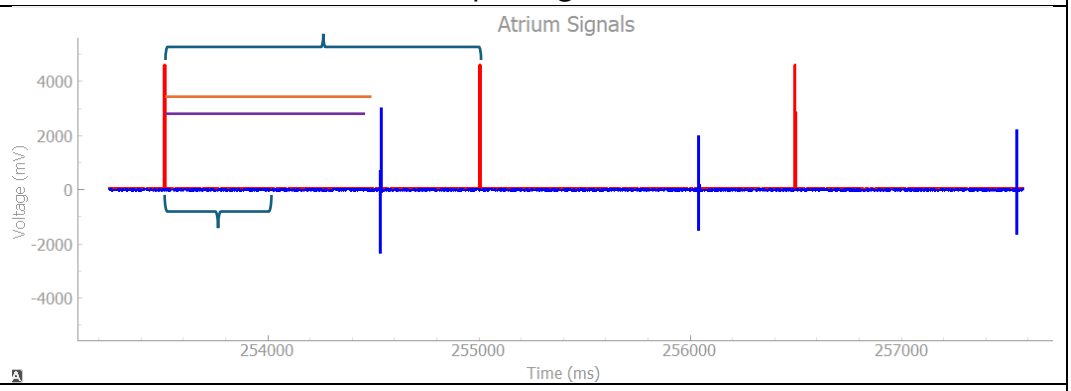
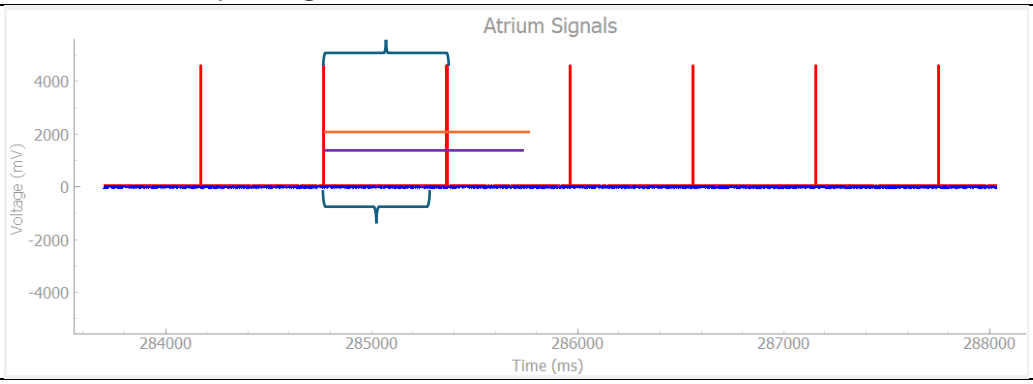
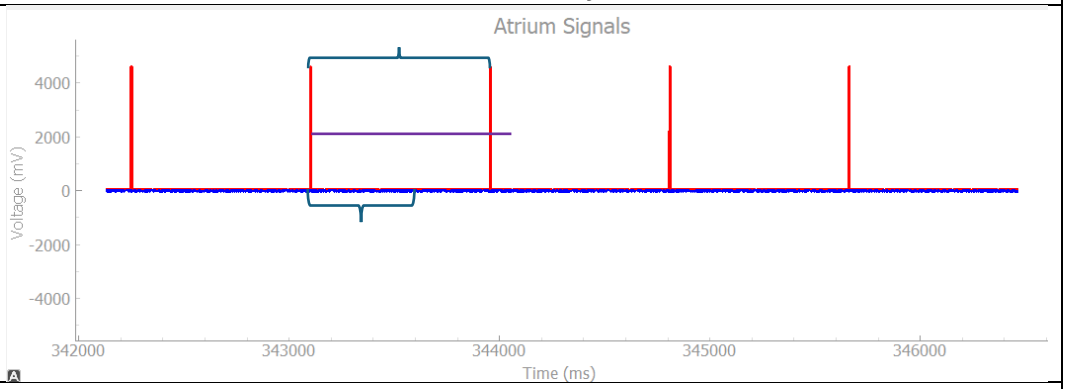
5.4.5 Heart View Tests

The mode was tested with heart view at various parameters to ensure functionality.

Table 31: Heart View results for AAI

Test Case	Result
CurrRate = 90 ppm APW = 1 ms ARP = 250 ms Hys = ON Natural ppm = 100	
Case 1: When the natural heart rate is greater than the current rate, pacing is inhibited.	
CurrRate = 90 ppm APW = 1 ms ARP = 250 ms Hys = ON Natural ppm = 70	
Case 2: When hysteresis is on, and the natural heart rate between the current rate and the LRL, pacing is inhibited	
CurrRate = 90 ppm APW = 1 ms ARP = 250 ms Hys = ON Natural ppm = 40	
Case 3: When hysteresis, is on, and the natural is below the LRL, pulses are sent to the atrium at the Cur Rate.	

CurrRate = 90 ppm APW = 1 ms ARP = 250 ms Hys = OFF Natural ppm = 100	
Case 4: When hysteresis is off, pacing is still inhibited when the natural heart rate exceeds the current rate.	
CurrRate = 90 ppm APW = 1 ms ARP = 250 ms Hys = OFF Natural ppm = 70	
Case 5: when the natural heart rate is below the current rate, the atrium is pulsed at the current rate.	
CurrRate = 60 ppm APW = 0.4 ms ARP = 150 ms Hys = ON Natural ppm = 80	
Case 6: at a lower current rate (60 bpm), pacing is inhibited when the natural heart rate is greater than the current rate.	
CurrRate = 60 ppm APW = 0.4 ms ARP = 150 ms Hys = ON Natural ppm = 40	

Case 7: At a lower current rate (60 bpm), a natural heart rate lower than the hysteresis rate causes pulsing.	
CurrRate = 60 ppm APW = 0.4 ms ARP = 500 ms Hys = ON Natural ppm = 40	
Case 8: at a greater atrial refractory period (500 ms), a natural heart rate below the hysteresis rate causes pulsing however, the window for detection decreases.	
CurrRate = 60 ppm APW = 0.4 ms ARP = 500 ms Hys = ON Natural ppm = 100	
Case 9: at a natural heart rate greater than the current rate, pacing is inhibited however the window for detection is very limited.	
CurrRate = 60 ppm APW = 0.4 ms ARP = 500 ms Hys = OFF Natural ppm = 70	
Case 10: When hysteresis is off, a natural heart rate greater than the current rate inhibits pulsing.	

5.4.6 Design Justification

AAI offers hysteresis functionality. However, the user has the option to toggle it on or off. Therefore, the mode should remain functional without hysteresis. Therefore, the 'SENSE' state functions independent of hysteresis. The 'HYSTERESIS' state includes added functionality which allows prolonged sensing upon a sensed natural atrial event.

This mode actively monitors the atrium to inhibit pacing if the heart is beating beyond a desired pace. Furthermore, discharging a capacitor to pulse is crucial as it protects the user from direct exposure to the power source. The capacitor can also be safely grounded to prevent further complications.

5.4.7 Requirements Changes

As of October 25th, 2024, no requirement changes have been made.

5.4.8 Design Decision Changes

1. Must check and modulate the atrial refractory period if it exceeds the msec duration of current rate.
2. Must implement safety checks for all input variables with a range including AS and APW.

5.5 Rate Adaptive Pacing

5.5.1 Description

Rate adaptive pacing increases or decreases an individual's pacing heart rate depending on the person's physical activity. For example, if a person is exercising which indicates that their body requires a greater blood flow, then the heart must be able to pump blood at a faster rate. Thus, if the activity level is above the activity level threshold, the pacing must increase accordingly over a set amount of time. However, if the person relaxes following an exercise session, their body must adapt to the lower activity level and gradually pump blood at a lower rate. This adaptivity must be reflected within the pacemaker's functionality which introduces the need for rate adaptive pacing modes such as VOOR, AOOR, VVIR, AAIR, and DDDR.

5.5.2 Requirements

1. Must take accelerometer data to calculate a desired heart rate.
2. Must pace at a desired heart rate that it must adapt to.
3. Must take a lower rate limit that it should not go below.
4. Must take a maximum sensor rate which it cannot exceed.
5. Must increase to the higher desired rate over a certain reaction time.
6. Must reduce to the lower desired rate over a certain recovery time.
7. Must not decrease or increase by a factor greater than rate smoothing.

5.5.3 Algorithm

A 3-axis accelerometer on the pacemaker pulse generator is used to measure an acceleration and is used to derive a desired heart rate proportional to physical activity. Given some accelerations, the pacemaker will first find the absolute value before taking a moving average that maps to an activity rate.

5.5.3.1 Equation

The activity rate is mapped to a desired heart rate through an exponential relationship given by the equation:

$$Desired\ Rate = LRL + (MSR - LRL) \times \left(1 - e^{-k \times \max(0,\ activity - AThresh \times 3)}\right)$$

where

$$k = 0.0876 \times RF + 0.1$$

5.5.3.2 Variables and Constants for Desired Heart Rate

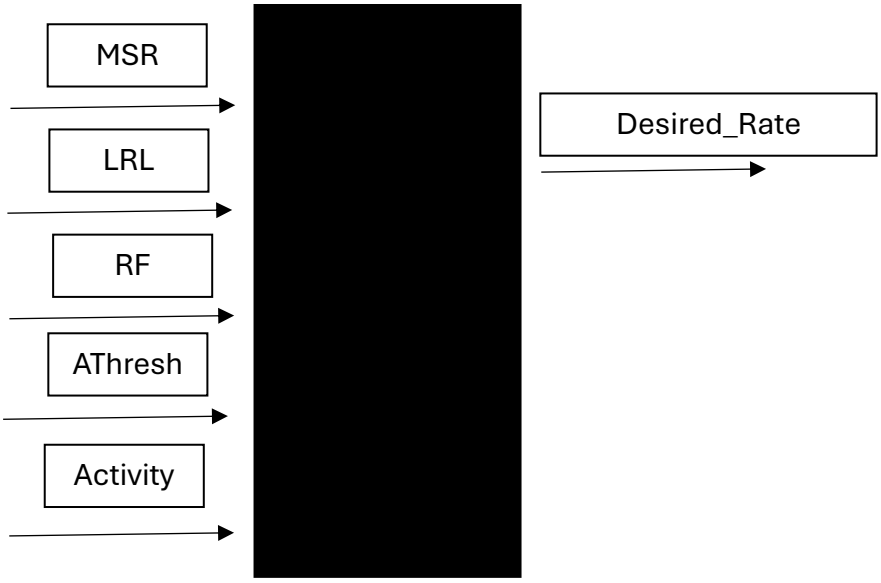


Figure 11: Black Box Diagram for Desired Heart Rate.

5.5.3.3 Monitored Variables

Table 32: Monitored Variables for Desired Heart Rate.

Variable	Name	Range	Nominal Value	Description
Physical Activity	Activity	0 – 50	-	Calculated physical activity based on acceleration.

5.5.3.4 Controlled Variables

Table 33: Controlled Variables for Desired Heart Rate.

Variable	Name	Range	Nominal Value	Description
Desired Rate	Desired_Rate	LRL-MSR ppm	-	Calculated Desired Rate based on activity level and activity threshold.

5.5.3.5 Programmable Parameters

Table 34: Programmable Parameters for VOOR.

Variable	Name	Range	Nominal Value	Description
Lower Rate Limit	LRL	50-90 ppm	60 ppm	Minimum ppm at which pulses are delivered in the absence of sensed activity

Maximum Sensor Rate	MSR	50-175 ppm	120 ppm	Maximum pacing rate due to activity accelerometer control.
Response Factor	RF	1 – 16	8	Corresponds to increment of change in desired heart rate.
Activity Threshold	AThresh	1 (V-Low) – 7 (V-High)	4 (Med)	The value the accelerometer data need to exceed before the pacemaker's rate is changed.

5.5.3.6 Simulink Explanation

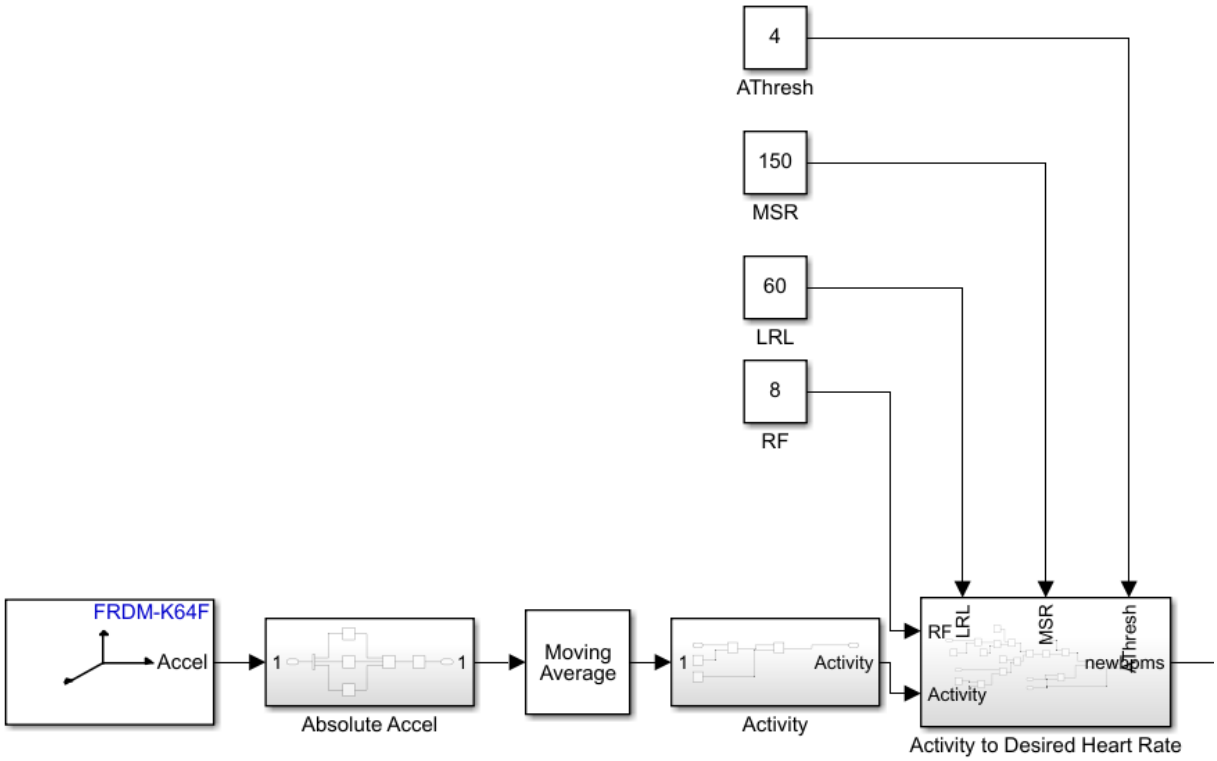
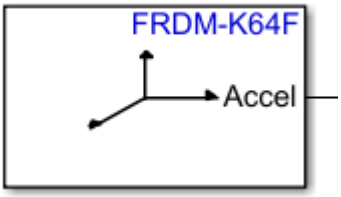
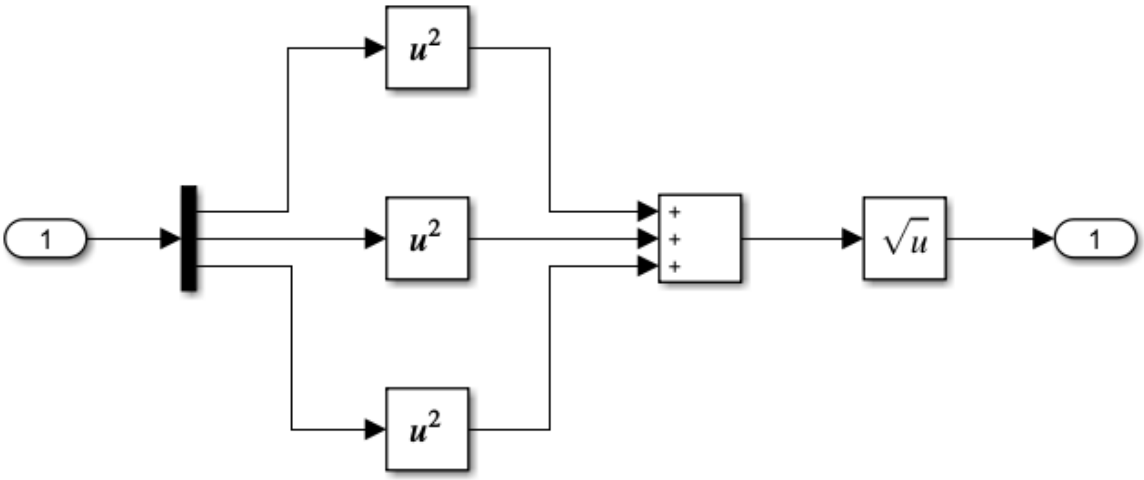
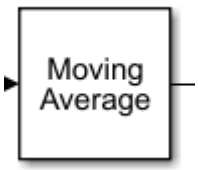


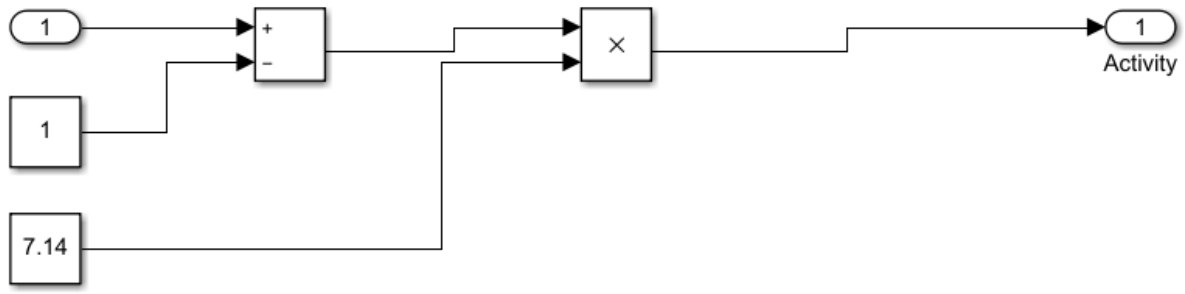
Figure 12: Simulink diagram for the Desired Heart Rate.

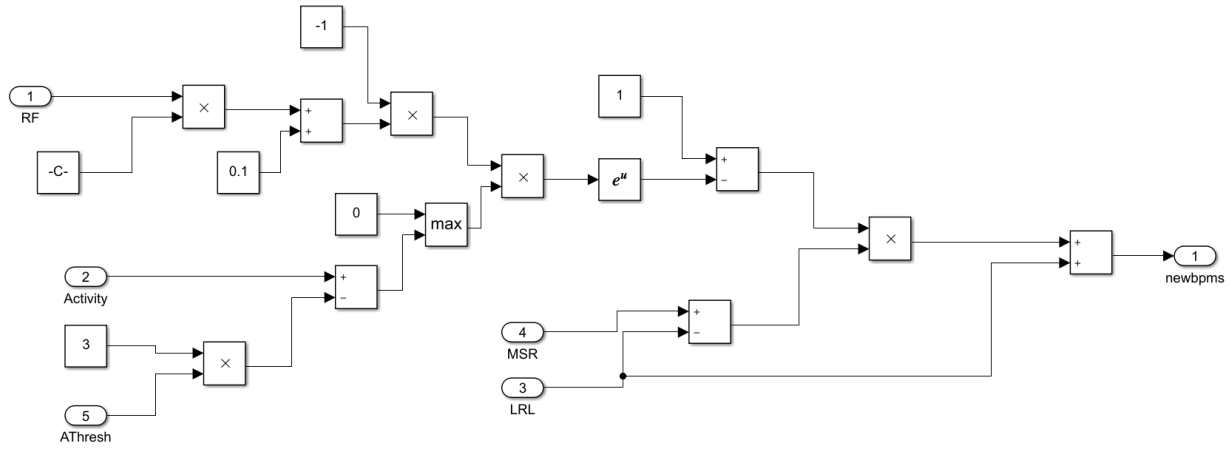
5.5.3.7 States

Table 35: Breakdown of state flow chart states and transitions.

Accel Block

This block reads acceleration data from the onboard accelerometer up to ±4g.
Absolute Accel

The Absolute Accel subsystem takes the accelerometer data from the Accel Block, separates the x, y, and z parameters, then takes the sum of the squares before taking the square root to get the absolute magnitude of acceleration.

Moving Average

The Moving Average Block takes the result from the Absolute Accel and using a sliding window of length 200, calculates an average acceleration.

Activity

The Activity Block takes the data from the Moving Average Block and maps the average acceleration to an activity. Knowing that the Accelerometer Block can read up to ±4g, the maximum that the Absolute Accel Block can output would be: $$\sqrt{3 \times 4^2} = 6.928 \approx 7$$ However, even if the pacemaker is not moving, the minimum value from the Absolute Accel Block is 1, so subtracting 1 from the input is first needed. To map the accelerometer data to an activity on a scale of 0 – 50, the linear equation is made: $$activity = 7.14 \times accel$$ This Activity Block will then output an activity between 0 – 50.

Activity to Desired Heart Rate

Taking inputs of RF, Activity, LRL, MSR and AThresh, the desired heart rate is calculated from the equation derived in **5.5.3.1**.

5.5.4 Variables and Constants for Adaptive Pacing

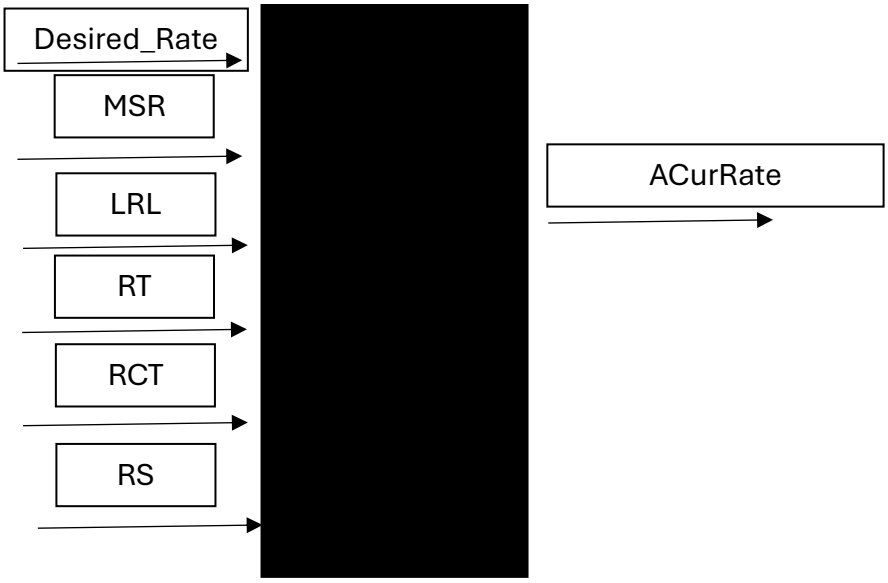


Figure 13: Black Box Diagram for Rate-Adaptive Pacing

5.5.4.1 Monitored Variables

Table 36: Monitored Variables for Rate Adaptive Pacing.

Variable	Name	Range	Nominal Value	Description
Desired Rate	Desired_Rate	LRL-MSR ppm	-	Calculated Desired Rate based on activity level and activity threshold.

5.5.4.2 Controlled Variables

Table 37: Controlled Variables for Rate Adaptive Pacing.

Variable	Name	Range	Nominal Value	Description
Adapted Current Rate	ACurRate or CR	LRL-MSR ppm	-	Rate to pace at when the pacemaker is adapting to activity.

5.5.4.3 Programmable Parameters

Table 38: Programmable Parameters for Rate Adaptive Pacing.

Variable	Name	Range	Nominal Value	Description
Lower Rate Limit	LRL	50-90 ppm	60 ppm	Minimum ppm at which pulses are delivered in the absence of sensed activity

Maximum Sensor Rate	MSR	50-175 ppm	120 ppm	Maximum pacing rate due to activity accelerometer control.
Reaction Time	RT	10-50 sec	30 sec	Time for the pacemaker to go from the lower rate limit to the maximum sensor rate when activity is sensed.
Recovery Time	RCT	2-16 min	5 min	Time for the pacemaker to go from the maximum sensor rate to the lower rate limit when activity is less than the threshold.
Rate Smoothing	RS	Off (0), 3, 6, 9, 12, 15, 18, 21, 25%	Off (0)	The difference in the heart rate interval as a percentage of the previous heart rate interval.

5.5.5 Simulink Explanation

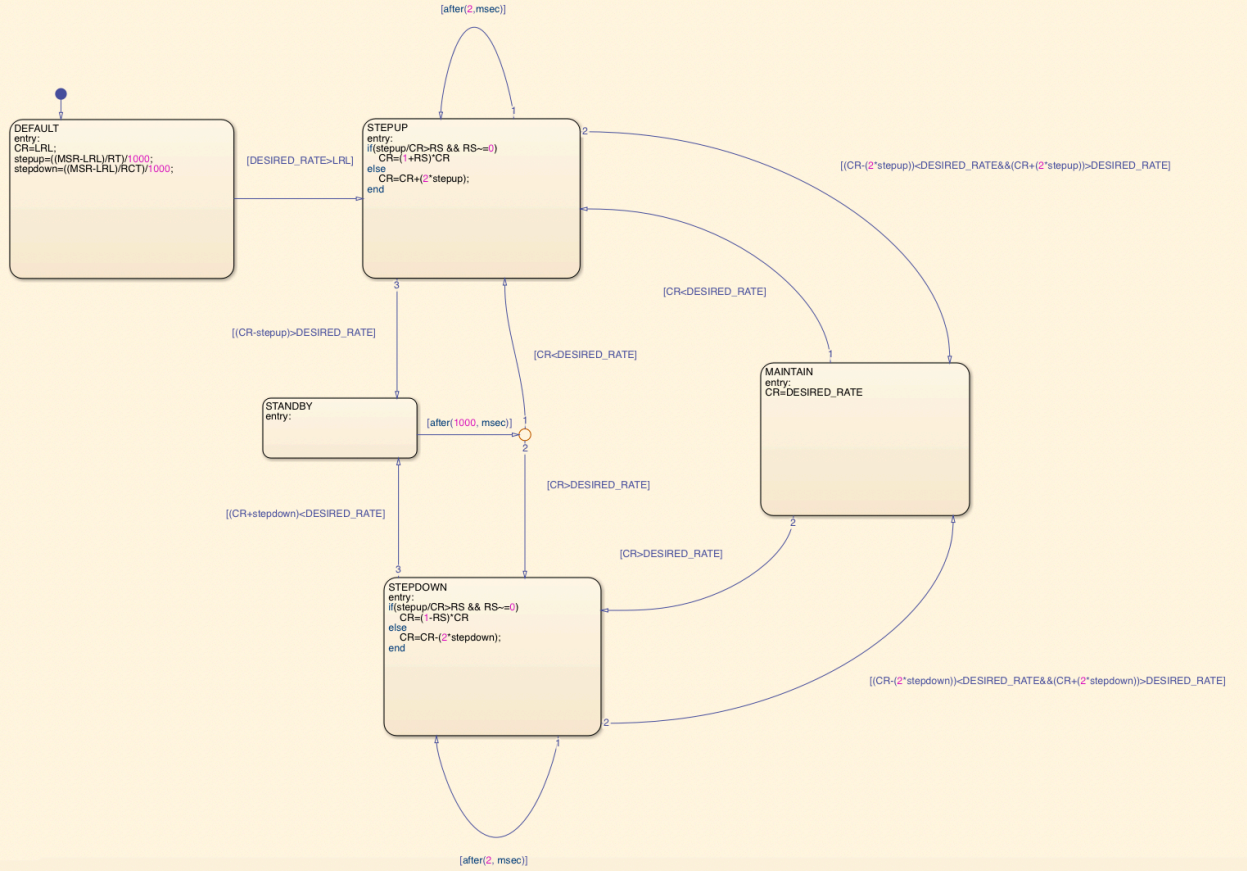
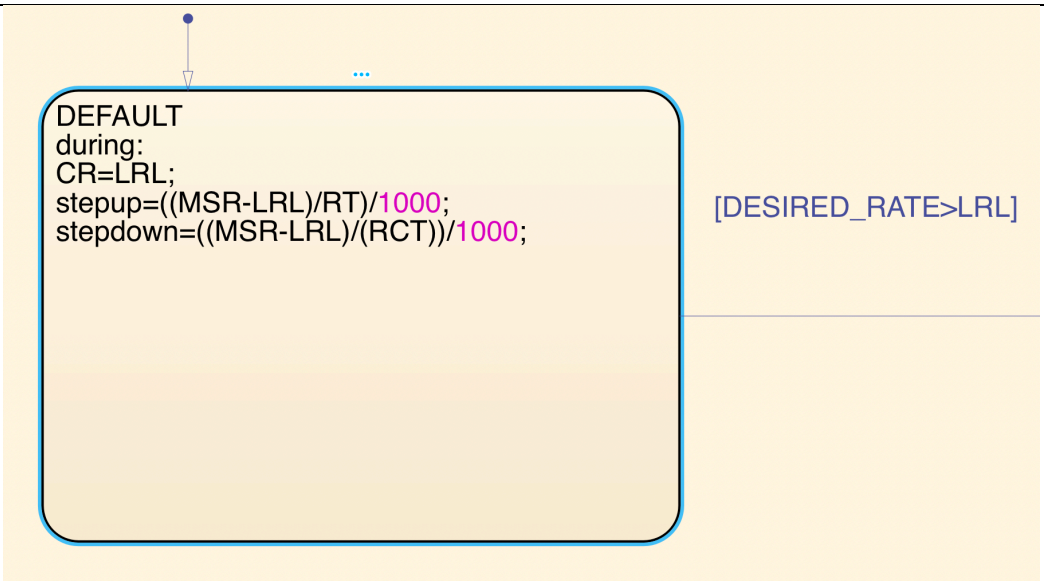
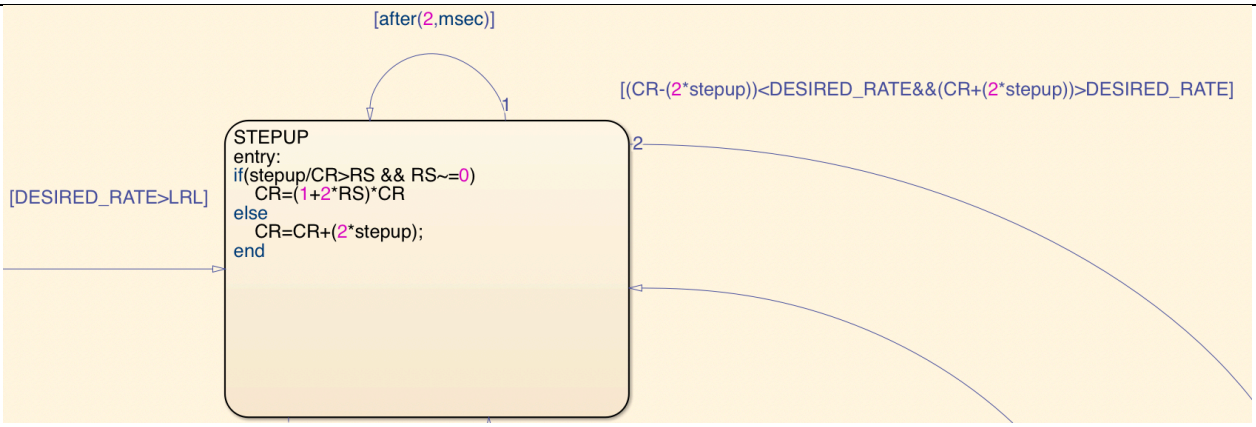


Figure 14: Simulink diagram for the Rate Adaptive Pacing.

5.5.5.1 States

Table 39: Breakdown of state flow chart states and transitions.

DEFAULT State

This state initializes the output of the state flow chart to be equal to the LRL. It also sets the amount that the pacemaker should step up and step down after every second. To calculate the step up, the difference between the maximum sensor and lower rate limit is taken and then divided by the programmable reaction time. This will calculate the increment through which the rate should be increased by. Similarly, the step down is the decrement that the rate must be adjusted by according to the desired rate derived from the accelerometer. Both these local variables are in units BPM/ms. $$Step\ up = \frac{MSR - LRL}{RT * 1000\frac{ms}{s}}$$ $$Step\ down = \frac{MSR - LRL}{RCT * 1000\frac{ms}{s}}$$ The pacemaker remains in this state until a desired rate is detected that is greater than the threshold which is LRL.
STEPUP State



This state increments the output adapted rate by the step up amount calculated in the previous state.

$$CR = CR + 2 * stepup$$

The step up is multiplied by two due to the fact that the incrementation occurs every 2 milliseconds.

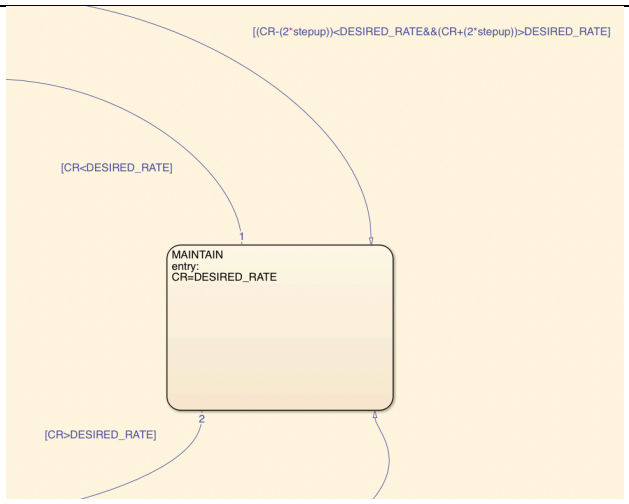
However, if the step up increment is a proportion of the previous adapted rate that is too high in comparison to the rate smoothing proportion, then the increment should be limited by the rate smoothing rate.

$$CR = (1 + 2 * RS) * CR$$

Note that the rate smoothing percentage is converted to be per millisecond in an outer subsystem. Since the heart rate is incremented every two milliseconds, then the RS must be per 2 milliseconds.

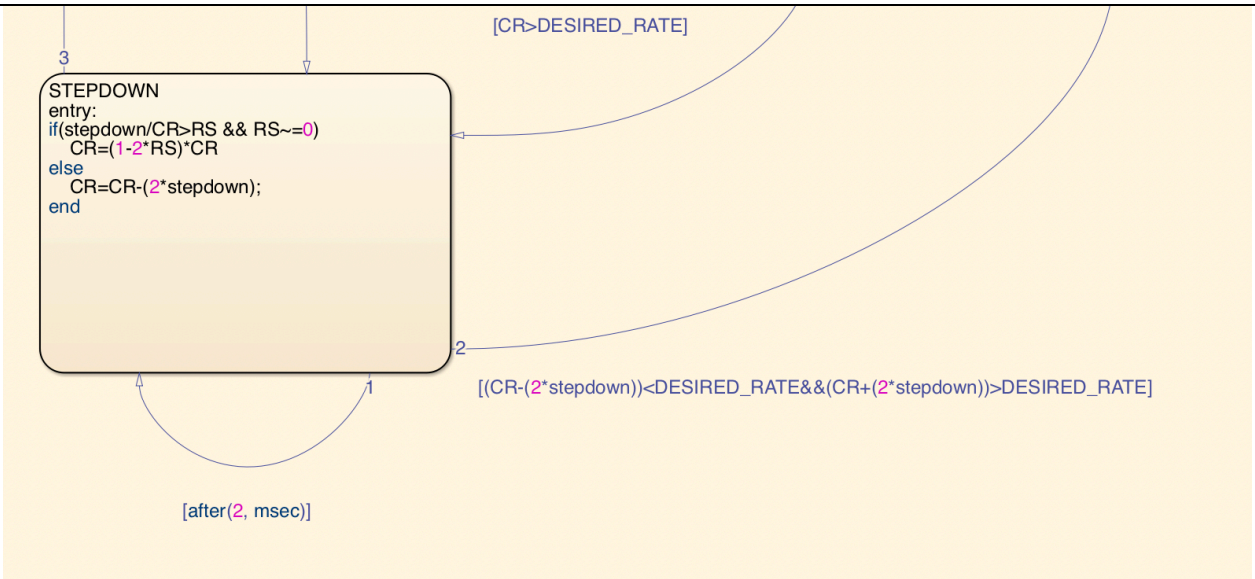
As long as the desired rate is constant, the pacemaker would continuously increase the current rate by the step up amount every 2 milliseconds. Once the current rate is within two step up increments of the desired rate then the desired rate is approximately reached. This is in place to reduce absolute equivalencies and take care of any overshoots or undershoots.

MAINTAIN State



Since the current adapted rate is approximately the desired rate, the two can be set equal to each other. If the desired rate is altered to be higher than the current adaptive rate due to an increase in activity level, then the pacemaker should return to the step up state. However, if the desired rate falls below the current rate, then the pacemaker should go into the step down state.

STEPDOWN State



If the new desired rate is below the current pacing rate, then the pacemaker should safely decrement the pace at which it is pacing. This is where the step down state is needed. This state behaves in a similar manner to the step up state. Within this state, if the proportion of decrease is less than the rate smoothing rate, then the current rate would be decremented by twice the step down. It is essential to decrement by twice the step down amount since the current rate is only ever decremented every 2 milliseconds.

$$CR = CR - 2 * stepdown$$

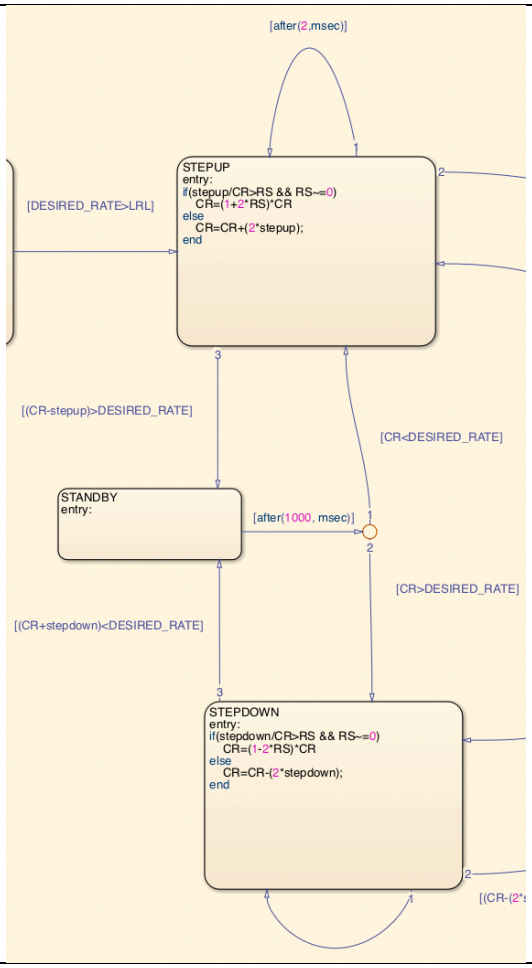
Alternatively, when the stepdown factor relative to the current rate is greater than the rate smoothing proportion, then the pacemaker will rely on a decrement value dependent on only the rate smoothing.

$$CR = (1 - 2 * RS) * CR$$

This also occurs every two milliseconds which is accounted for by the multiplication of the rate smoothing proportion by two.

When the current rate becomes within two stepdown decrements of the desired rate, the pacemaker will revert back to the MAINTAIN state.

STANDBY/DECISION State



Within the step up or step down states, the pacemaker should be able to adapt to sudden changes in pacing. For instance, if the pacemaker is within the step up state and it senses that the desired rate is lower than the current rate, then the pacemaker should go in a STANDBY state where it waits to verify that this change is not just a one time occurrence. It waits 1 second to validate that this change in desired rate is constant, then depending on what the current rate is. If the current rate is less than the desired rate, then it will go back to the STEPUP state. However, if the current rate is greater than the desired rate, then the

pacemaker should go into the STEPDOWN state. The same process should occur, if the pacemaker is initially in the STEPDOWN state and it receives a sudden increase in the desired rate.

This overall state flow outputs an adapted current rate that can be used in any of the rate adaptive modes.

5.5.6 Heart View Test

The following minimal test set ensures path coverage for the Simulink model.

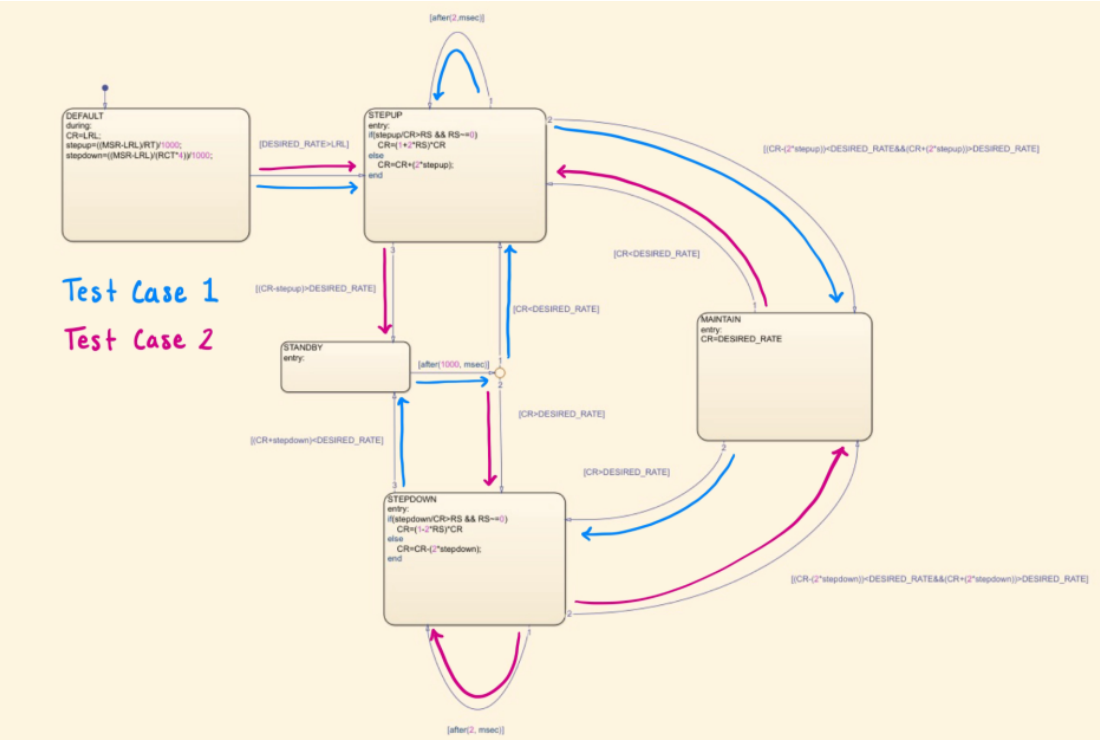
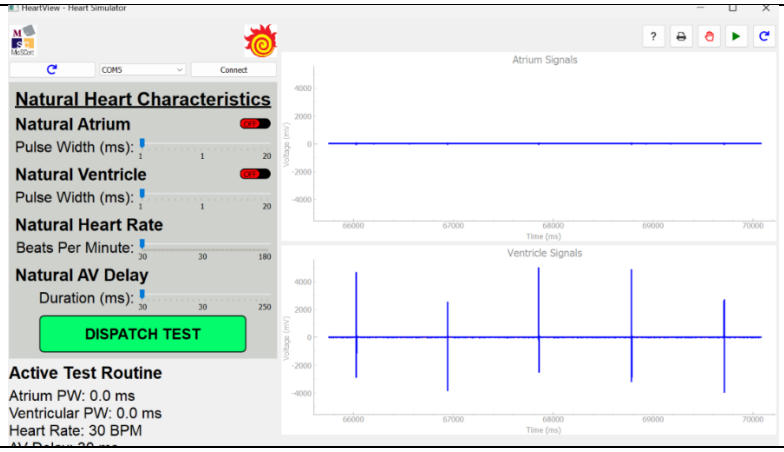
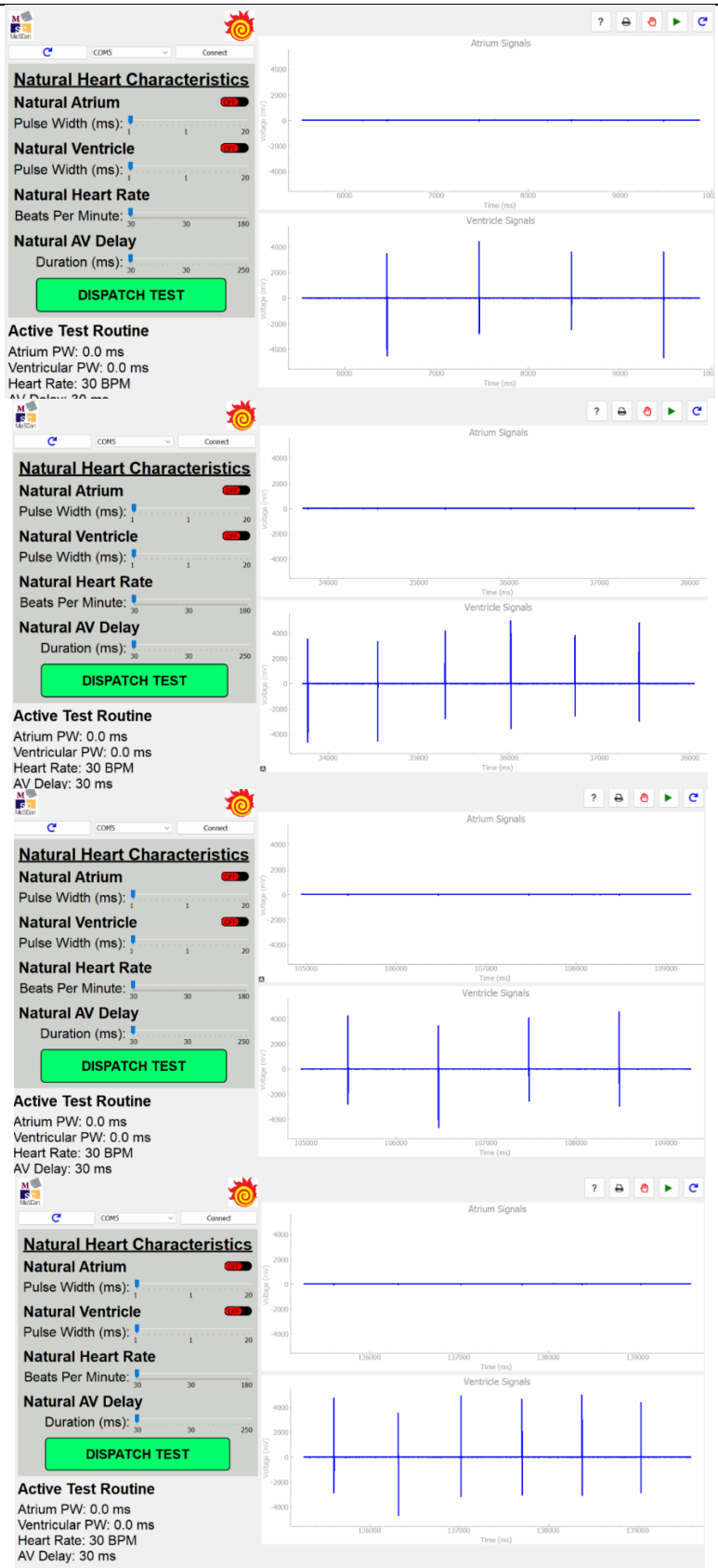


Figure 15: Test case coverage.

Table 40: Path coverage test for rate adaptive pacing.

Test Case	Result
Shake pacemaker to increase t > RT Stop shaking (wait) Shake again	

Case 1: Shake to increase activity to increase the pace to MSR, stop shaking to decrease the pace, increase the pace again. Behaves as expected.

Shake pacemaker to increase t < RT Stop shaking (wait) Shake again	

COM5
Connect
Natural Heart Characteristics
Natural Atrium
Pulse Width (ms):
Natural Ventricle
Pulse Width (ms):
Natural Heart Rate
Beats Per Minute:
Natural AV Delay
Duration (ms):
DISPATCH TEST
Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 30 BPM
AV Delay: 30 ms
Atrium Signals
Ventricle Signals
Voltage (mV)
Time (ms)

Case 2: Shake to increase activity to increase the pace, stop shaking to decrease the pace, increase the pace again. Behaves as expected.

5.5.7 Design Justification

From **5.5.3**, an algorithm was created with the equation from **5.5.3.1.** This equation is an exponential function that takes RF, Activity, LRL, MSR, and AThresh as parameters and is illustrated below.

Figure 16: Illustration of the equation for desired heart rate from **5.5.3.1.**

In terms of the adaptive rate pacing, the current design takes into account all the possible situations that occur during rate adaptive pacing. Within rate adaptive pacing, the pacing must increase if activity is greater than the threshold are detected. Additionally, this rate should be maintained until a change in activity is detected. If there is a decrease in activity,

the pacing should decrease to the lower rate limit. However, if any sudden increases or decreases in desired rate are detected then the pacemaker should look out for those changes, verify that they are constant then respond accordingly. This aligns with the requirement that heart's pacing rate must be able to adapt to changes in activity level.

5.5.8 Requirement Changes

1. Must be customizable for different age groups.
2. Must have sensor input that covers all range of motion.
3. Must account for uncertainty ranges within the motion sensors.

5.5.9 Design Decision Changes

For the rate adaptive sensing, the activity thresholds determined could be dependent on actual physical activity data. For instance, heart rate data could be gathered from a few individuals of different age groups. This will assist in making the pacemaker more customizable for people of different age groups. Moreover, as of right now, vigorous shaking must be applied to reach a noticeable increment in pacing. However, this may not be reflective of actual human activity. As well, an accelerometer placed only within the pacemaker may not be sufficient to cover all range of motion of various body parts. More sensors could be added within the pacemaker or in adjacent areas to ensure that all correct motion is being detected. Furthermore, a range of uncertainty could be derived for each sensor that could be incorporated in the mode to ensure the safety of rate incrementation.

5.6 VOOR

5.6.1 Description

VOOR must behave similarly to the VOO pacing. This means that it must also pace the ventricle regardless of the ventricular activity. The only difference is that this mode employs rate adaptive pacing. This means that the rate at which the ventricle is paced will increase or decrease depending on the current pacing rate and the rate that must be reached to accommodate the change in activity level.

5.6.2 Requirements

1. Must use an adapted pacing rate proportional to the physical activity to pace the ventricle.
2. Must follow all the other requirements described in section **5.1.2** of the VOO requirements.

5.6.3 Variables and Constants

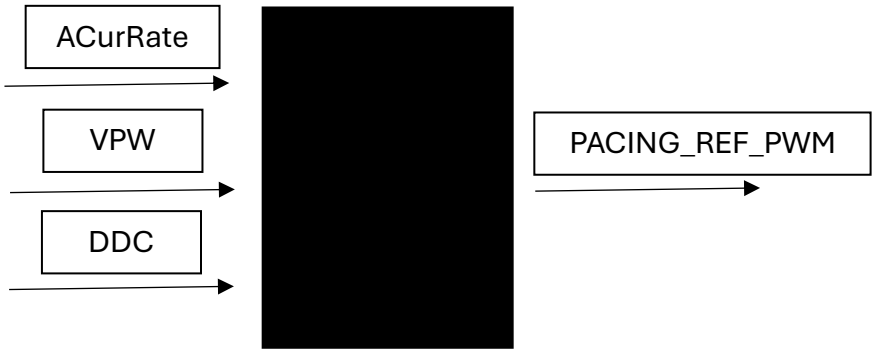


Figure 17: Black Box Diagram for VOOR

5.6.3.1 Monitored Variables

Table 41: Monitored Variables for VOOR.

Variable	Name	Range	Nominal Value	Description
Adapted Current Rate	ACurRate	LRL-MSR ppm	-	Adaptive current rate derived using physical activity.

The monitored variable shown above is in place of the current rate used to pace at in the VOO mode. However, all other monitored variables in the VOO mode are also used in the VOOR mode. Therefore, the rest of the monitored variables used in this mode are summarized in the VOO mode monitored variables in section **5.1.3.1**.

5.6.3.2 Controlled Variables

The controlled variables utilized in the VOOR mode are exactly the same as the VOO mode controlled variables in section **5.1.3.2**.

5.6.3.3 Programmable Parameters

Refer to the programmable parameter listed in the VOO mode section **5.1.3.3**.

5.6.4 Simulink Explanation

The Simulink model for the VOOR mode is exactly the same as the one utilized in the VOO mode. The only difference is that the transition between charging and discharging relies on a wait dependent on the adaptive pacing rate. For more details regarding the VOOR model refer to the Simulink model explanation in section **5.1.4.**

5.6.5 Design Justification

The adaptive rate pacing that this model follows is used to ensure that the ventricle is paced in accordance with the physical activity sensed. Refer to **5.1.5** for more information regarding the design justification of the VOO model.

5.7 AOOR

5.7.1 Description

The AOOR mode operates in a similar way to the AOO mode. However, it takes in the adapted current rate which is dependent on the physical activity conveyed by the accelerometer. This ensures that the pacing of the atrium is dynamic and adapts to the owner's physical activity.

5.7.2 Requirements

1. Must use an adapted pacing rate proportional to the physical activity to pace the atrium.
2. Must follow all the other requirements described in section **5.2.2** of the AOO requirements.

5.7.3 Variables and Constants

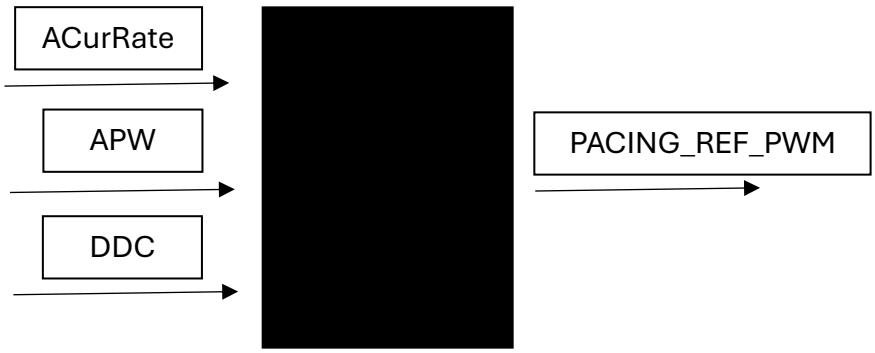


Figure 18: Black Box Diagram for AOOR

5.7.3.1 Monitored Variables

Table 42: Monitored Variables for AOOR

Variable	Name	Range	Nominal Value	Description
Adapted Current Rate	ACurRate	LRL-URL ppm	-	Adaptive current rate derived using physical activity.

The difference between AOOR and AOO is that AOOR utilizes the adapted current rate as opposed to the lower rate limit. This ensures that the atrium pacing dynamically changes with an increase or decrease in activity. All other monitored variables remained the same as the ones mentioned in the AOO mode. These variables can be found in section **5.2.3.1**.

5.7.3.2 Controlled Variables

The controlled variables utilized in the AOOR mode are exactly the same as the AOO mode controlled variables in section **5.2.3.2**.

5.7.3.3 Programmable Parameters

Refer to section **5.2.3.3** for an in depth explanation of the programmable parameters.

5.7.4 Simulink Explanation

This model is almost exactly the same as the AOO model. The only difference is the use of the adaptive rate pacing interval as a wait time between charging and discharging. For an in-depth explanation of the Simulink set up refer to section **5.2.4**.

5.7.5 Design Justification

The AOOR mode must be able to pace the atrium without considering the intrinsic atrium pulse. However, it should consider an increase or decrease in physical activity and adapt the current rate corresponding to the sensed activity level. The pacing aspect is better justified in section **5.2.5** of the AOO mode. However, the rate adaptive nature is better justified in section **5.5.7**.

5.8 VVIR

5.8.1 Description

The VVIR mode paces, senses, inhibits and rate adapts the ventricle. Essentially, this mode is identical to the VVI mode with the addition of rate adaptive pacing. This allows the VVI mode to dynamically adapt to fluctuations in physical activity that would require faster pacing of the ventricle.

5.8.2 Requirements

1. Must use an adapted pacing rate proportional to the physical activity to pace the ventricle.
2. Must follow all the other requirements described in section **5.3.2** of the VVI requirements.

5.8.3 Variables and Constants

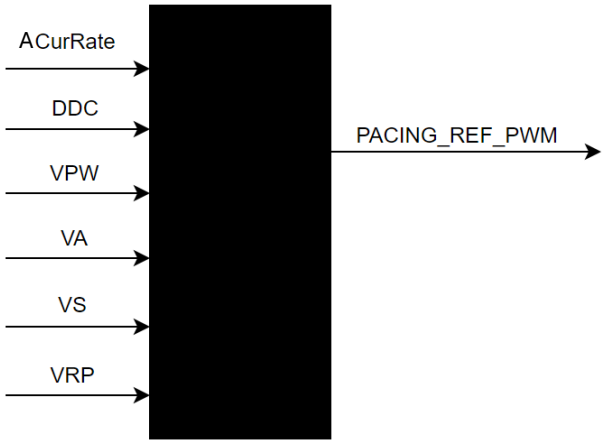


Figure 19: Black Box Diagram for VVIR

5.8.3.1 Monitored Variables

Table 43: Monitored Variables for VVIR.

Variable	Name	Range	Nominal Value	Description
Adapted Current Rate	ACurRate	LRL-URL ppm	-	Adaptive current rate derived using physical activity.

This mode differs from the VVI mode by utilizing the adapted pacing rate as opposed to the lower rate limit. The other monitored parameters within this mode could be found in section **5.3.3.1** of the VVI mode.

5.8.3.2 Programmable Parameters

The programmable parameters of this mode remain unchanged when compared to the VVI mode. For more information regarding the programmable parameters, refer to section **5.3.3.2**.

5.8.4 Simulink Explanation

The only difference between the VVIR Simulink model and the VVI Simulink model is the use of the adaptive rate pacing variable. Thus, to understand the logic behind the Simulink model, refer to section **5.3.4.**

5.8.5 Design Justification

This model achieves the pacing, sensing and inhibiting requirements of the VVIR mode by having states that take care of all three scenarios. This adherence to the requirements is better described in section **5.3.7** of the VVI mode. Moreover, this model takes into account rate adaptive pacing since it uses the adapted current rate to dynamically alter the pacing rate in accordance with the physical activity. As for the design justification of the rate adaptive pacing, it is detailed further in section **5.5.7**.

5.9 AAIR

5.9.1 Description

The AAIR mode operates in a similar manner tp the AAI mode by pacing, sensing, and inhibiting the atrium. However, it has the added functionality of regulating the heart rate based on the physical activity detected. This allows the mode to adapt to changes in activity that may warrant a change in heart rate.

5.9.2 Requirements

1. Must use an adapted pacing rate proportional to the physical activity to pace the ventricle.
2. Must follow all the other requirements described in section **5.4.2** of the AAI requirements.

5.9.3 Variables and Constants

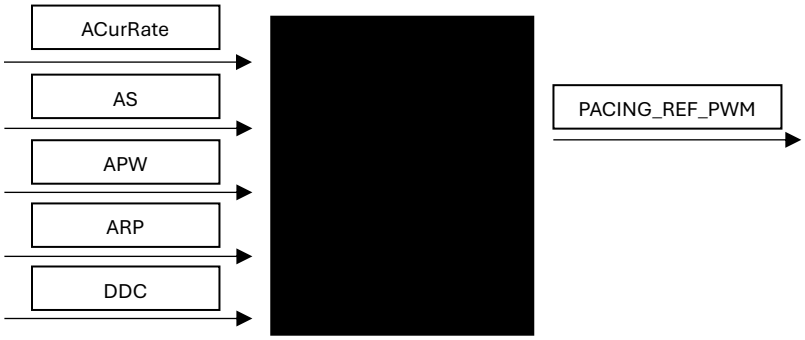


Figure 20: Black Box Diagram for AAIR

5.9.3.1 Monitored Variables

Table 44: Monitored Variables for AAIR

Variable	Name	Range	Nominal Value	Description
Adapted Current Rate	ACurRate	LRL-URL ppm	-	Adaptive current rate derived using physical activity.

The AAIR mode utilizes the adapted current rate as an interval for the cardiac cycles. That is the only differentiator between the AAIR and AAI mode. Thus, the rest of the monitored variables match the ones mentioned in section **5.4.3.1** of the AAI mode.

5.9.3.2 Control Variables

The control variables remain the same as the AAI mode which are elaborated on in section **5.4.3.2.**

5.9.3.3 Programmable Parameters

The programmable parameters of the AAIR mode match the programmable parameters of the AAI mode which are mentioned in section **5.4.3.3.**

5.9.4 Simulink Explanation

The Simulink model for the AAIR model closely resembles the AAI Simulink model. The only difference is that the cardiac cycle interval is the adapted pacing rate. For more information regarding the logic of the Simulink model, refer to section **5.4.4.**

5.9.5 Design Justification

This design takes care of the pacing, sensing, inhibiting and rate adaptive functionalities of the AAIR mode. Essentially, the states shown in the Simulink showcase the logic utilized to pace, sense and inhibit the atrium. An in-depth justification of the design is found in section **5.4.6.** As for the adaptive pacing rate, its design justification is detailed in section **5.5.7** of the rate adaptive pacing.

5.10 DDDR

5.10.1 Description

The DDDR mode's functionality is to pace, sense and inhibit both the ventricular and atrial chambers of the heart. Moreover, this mode is rate modulated. This indicates that with increased physical activity, the pacemaker must increment the pacing of both chambers and then decrement them whenever the physical activity subsides.

5.10.2 Requirements

1. Must pace the atrium and ventricle using a rate that is dependent on physical activity.
2. Must sense the atrium and ventricle.
3. Must inhibit the atrium or ventricle when the natural heart chambers events are sensed.
4. Must have a delay between the atrium and ventricle pacing.

5.10.3 Variables and Constants

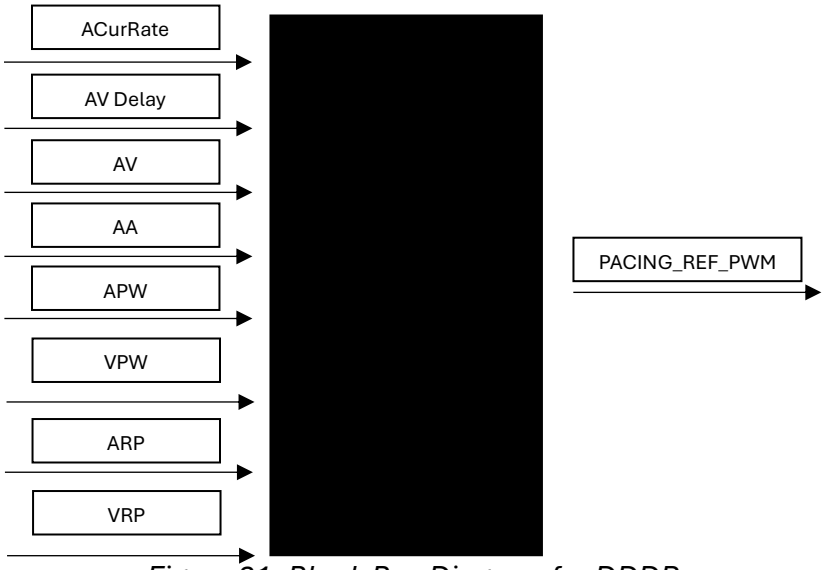


Figure 21: Black Box Diagram for DDDR

When comparing this mode to the other modes, the only additional input is the AV Delay which is a controlled variable.

Table 45: Monitored Variables for DDDR

Variable	Name	Range	Nominal Value	Description
AV_Delay	Atrioventricular Delay	70-300 ms	200 ms	This is the offset between an atrial and ventricular event.

5.10.4 Simulink Explanation

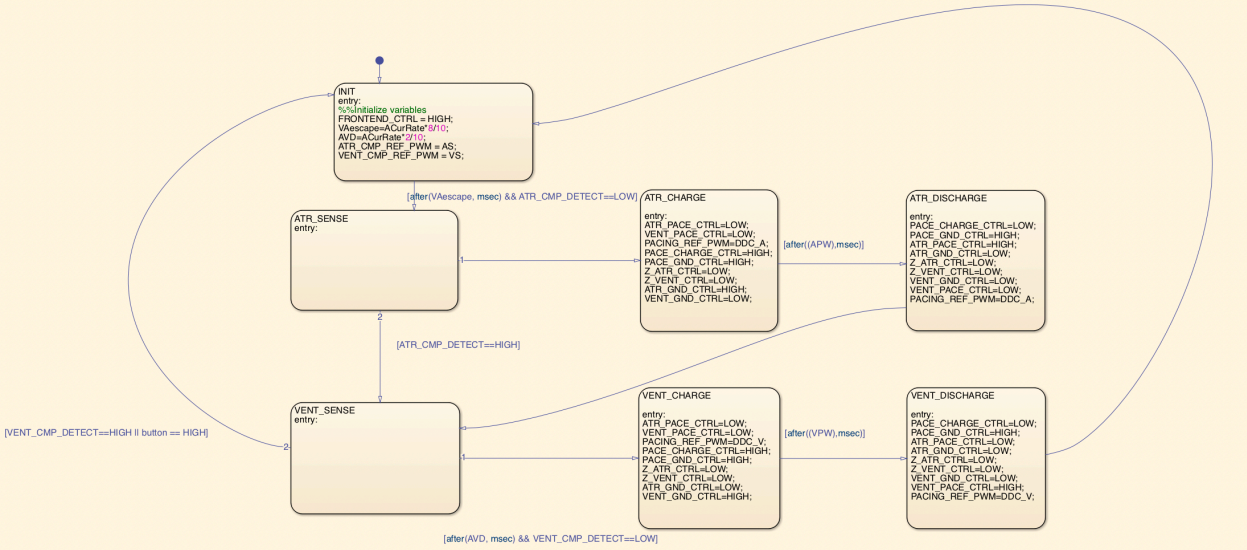


Figure 22: Simulink Model of DDDR

For this mode, five scenarios were identified. If no atrial or ventricular events are sensed, then the atrium must first be paced and then after the AV delay period, the ventricle must be paced.

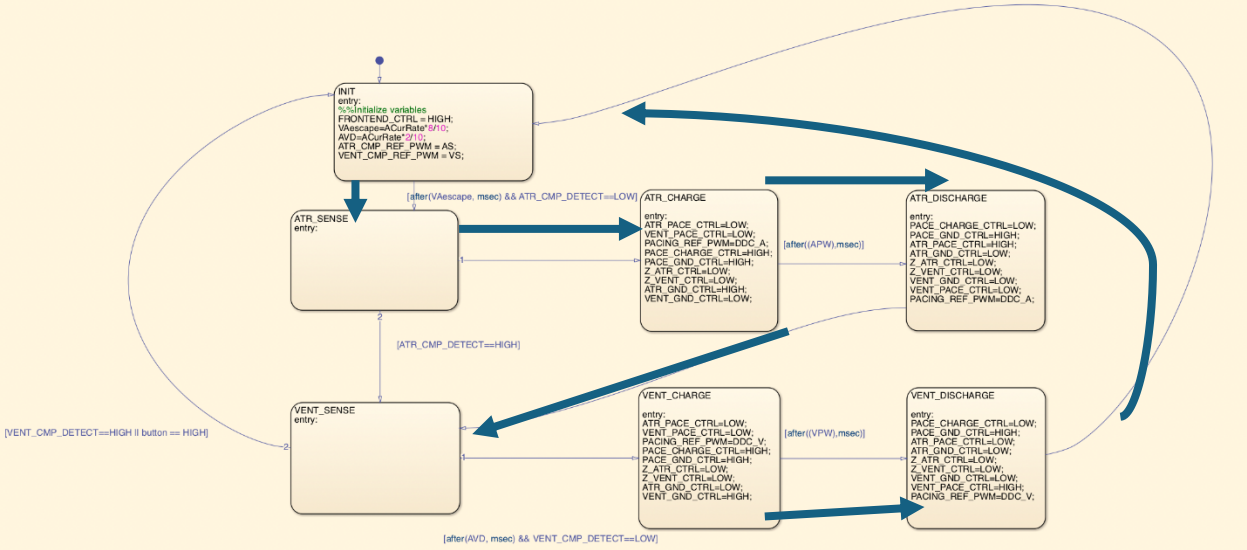


Figure 23: First Scenario.

Another case would be where an atrial event is not sensed, so an atrial pace is delivered but a ventricular event is sensed.

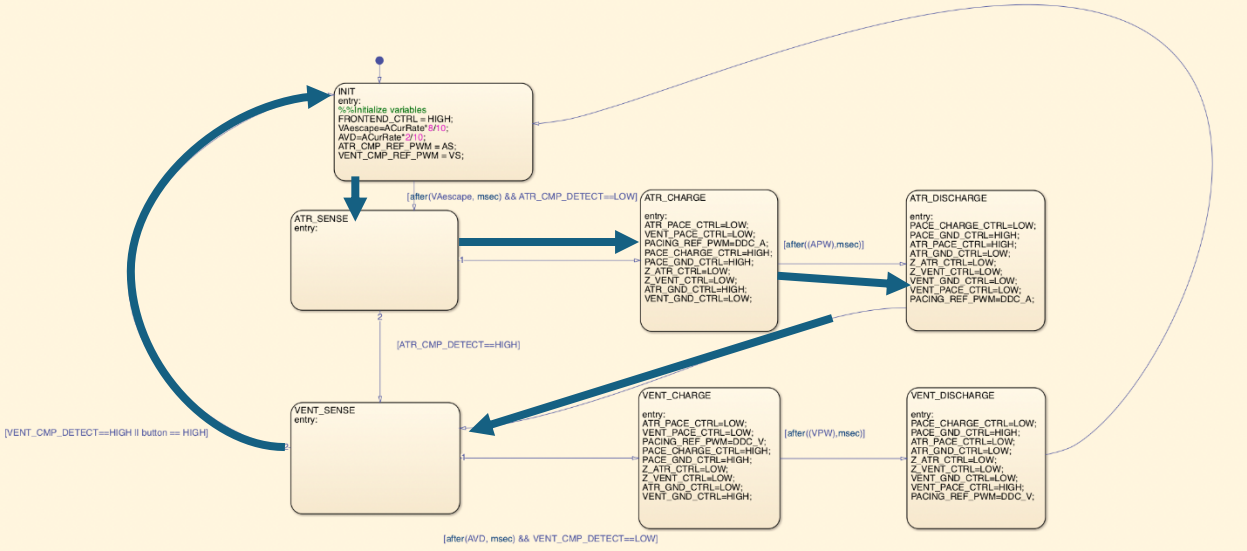


Figure 24: Second Scenario.

Alternatively, an atrial event could be sensed then following the delay a ventricular event is not sensed which results in an ventricular pace being delivered.

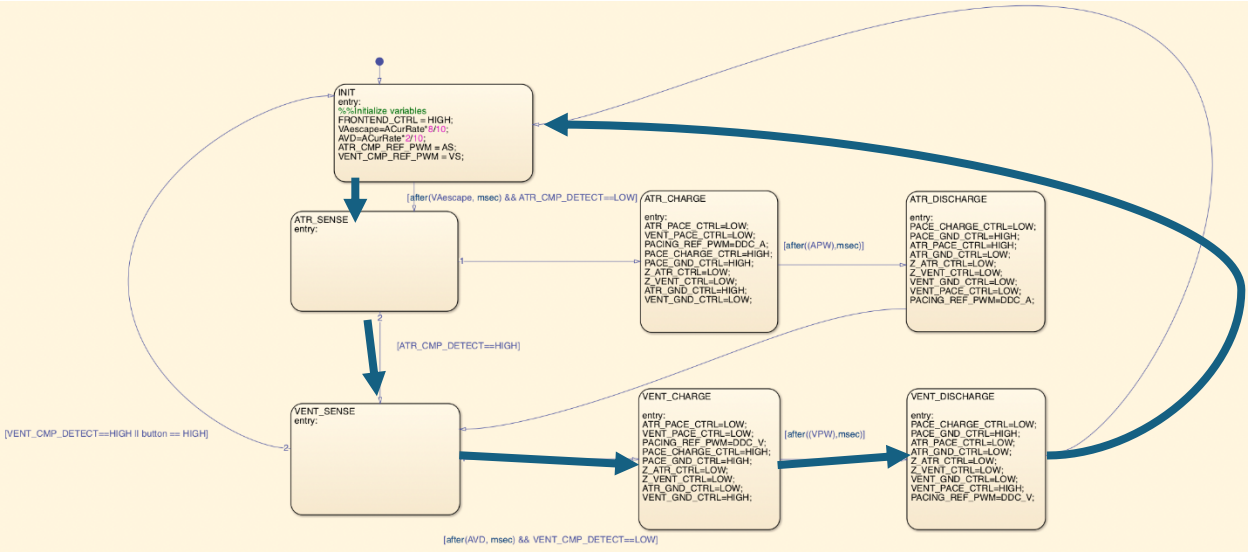


Figure 25: Third Scenario.

If the person's heart delivers both atrial and ventricular paces within the sensing durations, the pacemaker should skip pacing and go back into the sensing states.

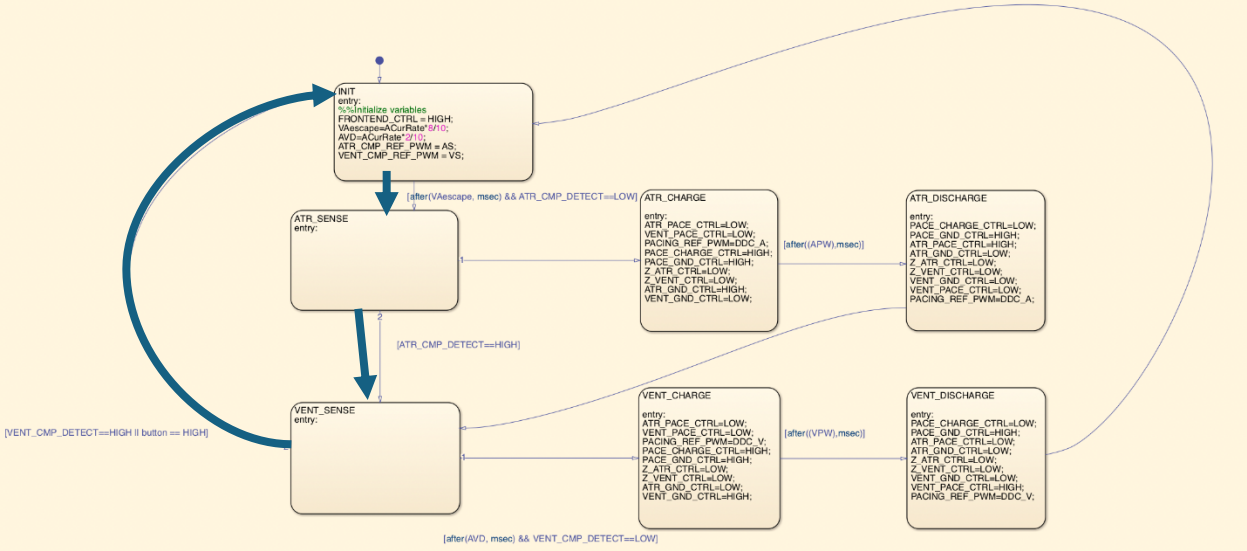


Figure 26: Fourth Scenario.

The final situation is when an increase in physical activity is sensed, then the adapted current rate must be updated accordingly. Moreover, this mode inhibits ventricular pacing when a button is pressed.

5.10.5 Design Justification

Since this model captures all possible combinations of events in the atrium and ventricle, it ensures comprehensive functionality. The inclusion of physical activity sensing allows the pacemaker to adapt to the patient's metabolic need. Moreover, this model ensures that the pacing of both chambers only occurs when they are not sensed. However, if they are sensed then it refrains from sending a pulse. This ensures that the pacemaker allows the heart to perform its intrinsic activity without being shocked when it is not necessary.

5.11 Serial Communication in Simulink

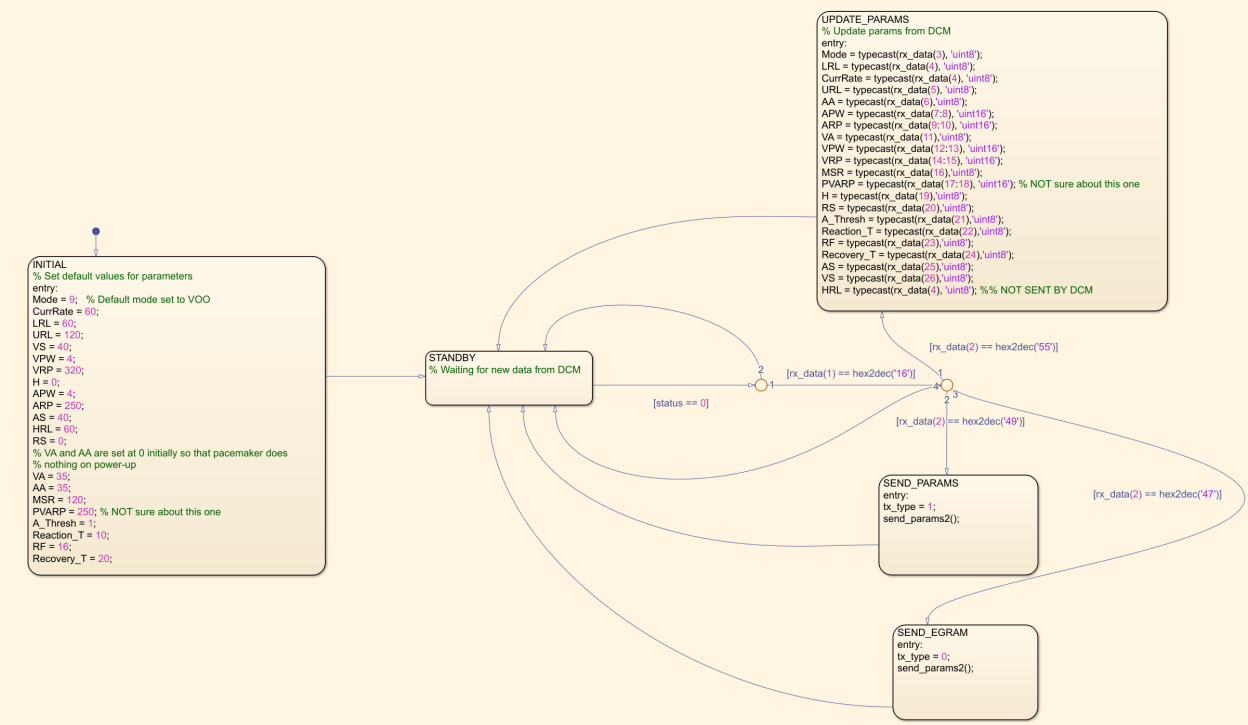


Figure 27: Stateflow that handles reading data sent from DCM via serial communication

An FSM is used to continuously check and respond to the data sent through UART. The FSM starts in the INITIAL state, setting initial values to the parameters required for each pacemaker mode. The pacemaker then waits in the STANDBY state until the status signal from the "Serial Receive" block goes to a 0, indicating that a new data packet is received. The first byte is read to ensure that the sync byte is correct. If the byte is 0x16, the second byte is checked for the function code. Depending on the function code, different actions are executed in the separate states defined. The UPDATE_PARAMS state updates the pacemaker parameters by parsing and typecasting the received data and returns to the STANDBY state. The SEND_PARAMS state sets tx_type = 1, executes the send_params() function, and returns to the STANDBY state. The SEND_EGRAM state sets tx_type = 0, executes the send_params() function, and returns to the STANDBY state. If either the sync or function code bytes are incorrect, the FSM returns to the STANDBY state to receive a new message.

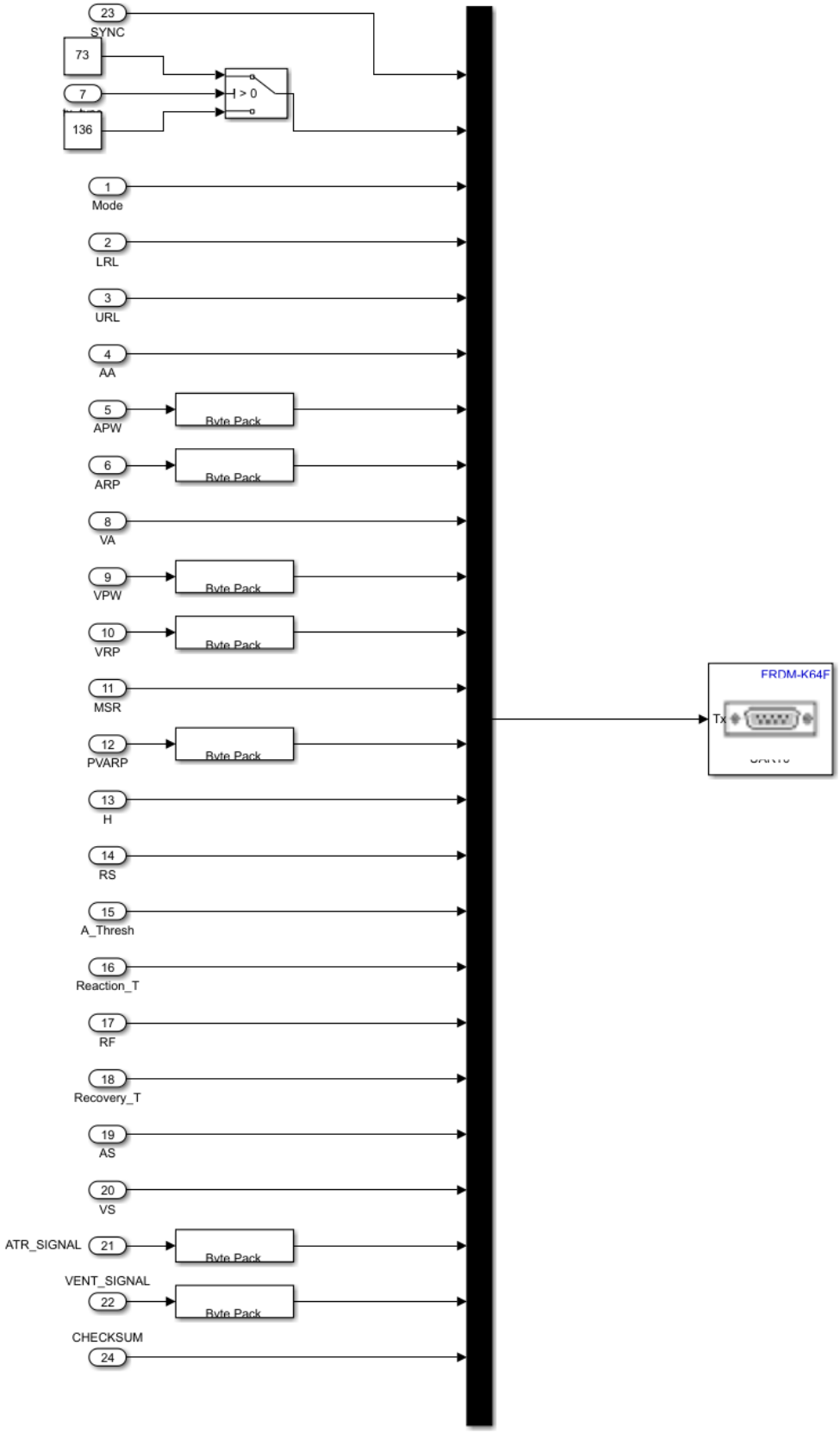


Figure 28: send_params() function

The send_params() function is used to send pacemaker parameters back to the DCM. The function consists of using a Mux block to combine all the parameters in order from top to bottom into a singular vector array. As the "Mux" block requires all inputs to be of the same data type, all inputs are set to uint8. As many of the parameters are already in uint8, other parameters such as the atrial refractory period (ARP) are converted from their respective data type to uint8 using the “Byte Pack” block. As the same function is used to send parameters and egram data, different function codes must be sent back to the DCM for validation. Therefore, a “Switch” block is used to send different function codes depending on the value of tx_type. If tx_type is 1, the pacemaker sends 73 (0x49) as the function code as the SEND_PARAMS state invoked the function. Otherwise, the pacemakers send 136 (0x88) as the function code as the SEND_EGRAM state invokes the function.

5.12 Safety

For a safety critical system, it is crucial to utilize redundancy, diversity, and defense-in-depth (DiD) to mitigate hazards to the greatest extent possible. These checks, along with the safety checks in DCM reduce the likelihood of invalid parameters falling through.
The following parameter safety checks have been implemented in Simulink:

Figure 29: Safety checks for pacemaker parameters

5.13 Design Justification

5.13.1 Testing

Table 46: Test Cases and Results for MODE.

Test Case	Result

MODE = 1 || MODE = 0

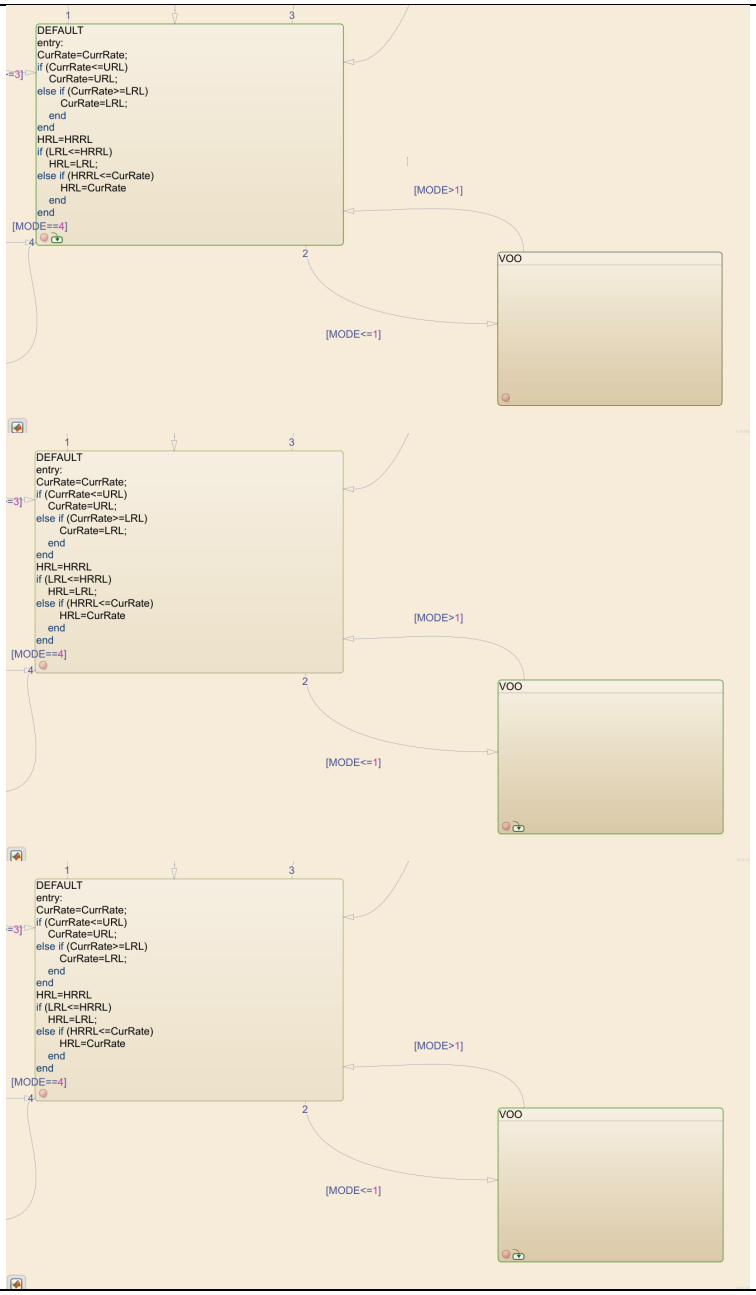


Fig. A

Fig. B

Fig. C

Test to go to VOO

When the MODE = 1, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the VOO state. The exact same thing happens when MODE = 0, illustrated in Fig. C. Therefore, the mode change into VOO with MODE = 1 and MODE = 0 works as intended.

MODE = 2

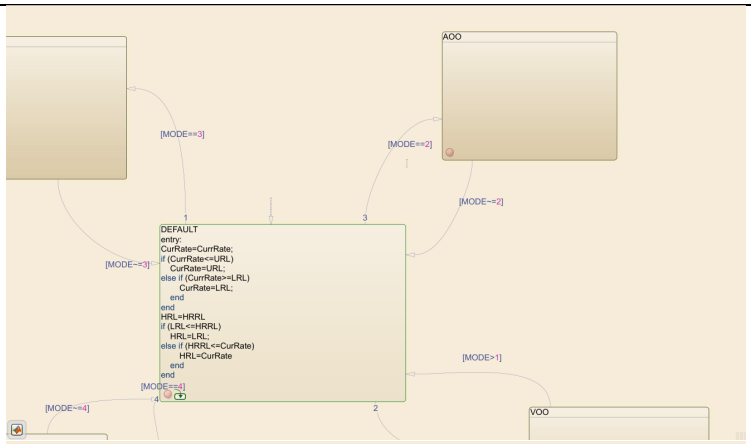


Fig. A

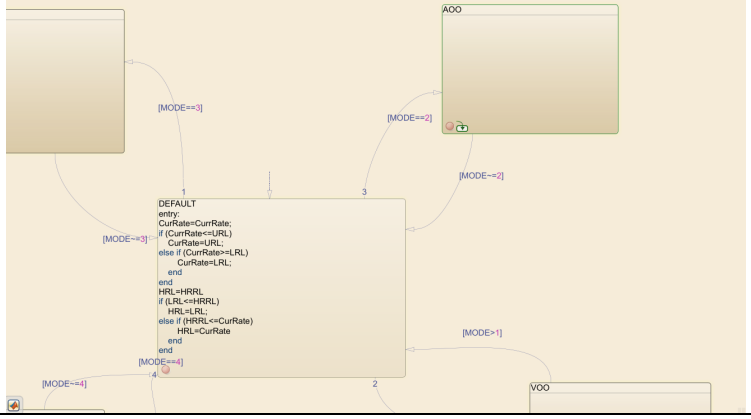


Fig. B

Test to go to AOO

When the MODE = 2, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the AOO state. Therefore, the mode change into AOO with MODE = 2 works as intended.

MODE = 3

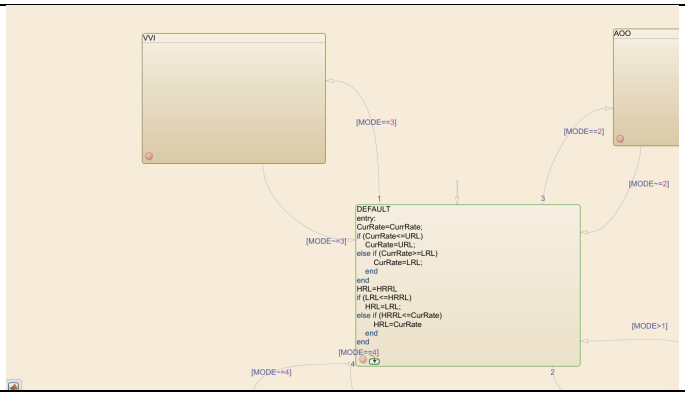


Fig. A

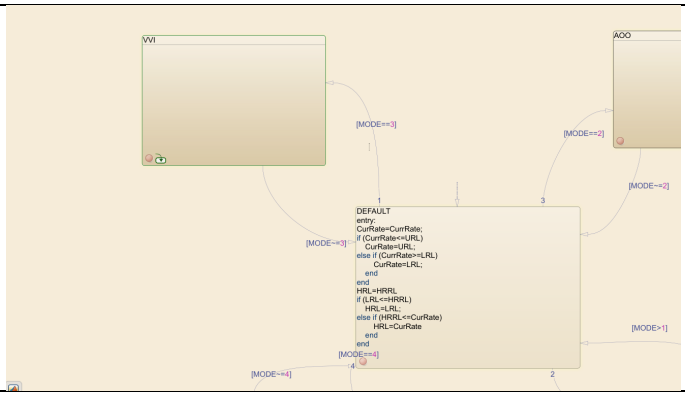


Fig. B

Test to go to VVI

When the MODE = 3, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the VVI state. Therefore, the mode change into VVI with MODE = 3 works as intended.

MODE = 4

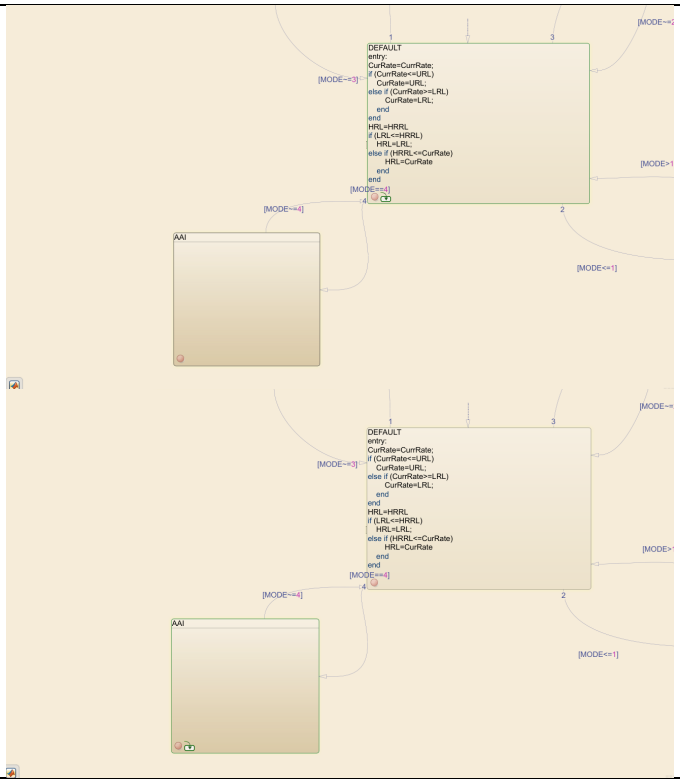


Fig. A

Fig. B

Test to go to AAI

When the MODE = 4, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the AAI state. Therefore, the mode change into AAI with MODE = 4 works as intended.

MODE = 5

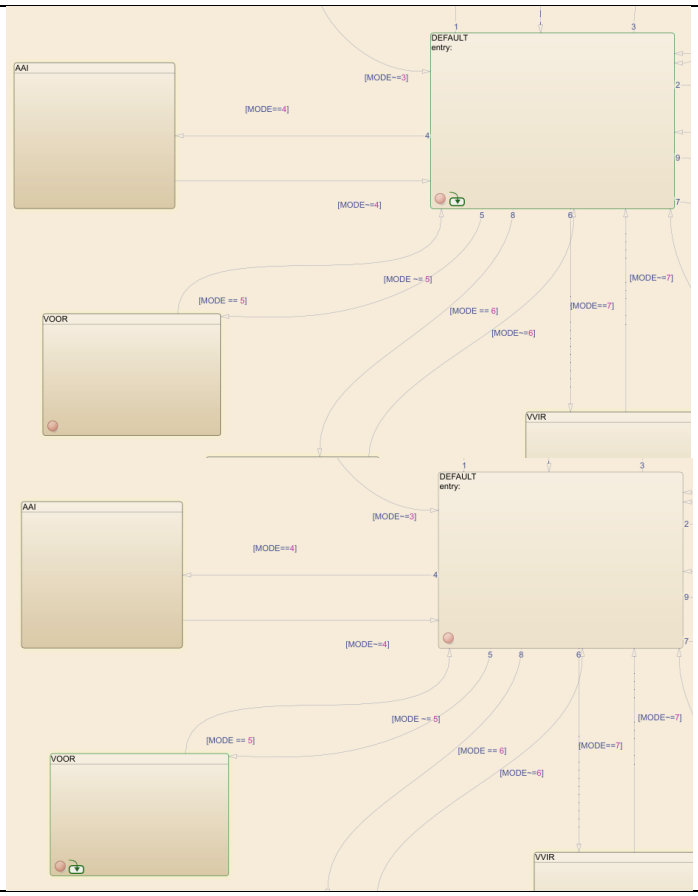


Fig. A

Fig. B

When the MODE = 5, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the VOOR state. Therefore, the mode change into VOOR with MODE = 5 works as intended.

MODE = 6

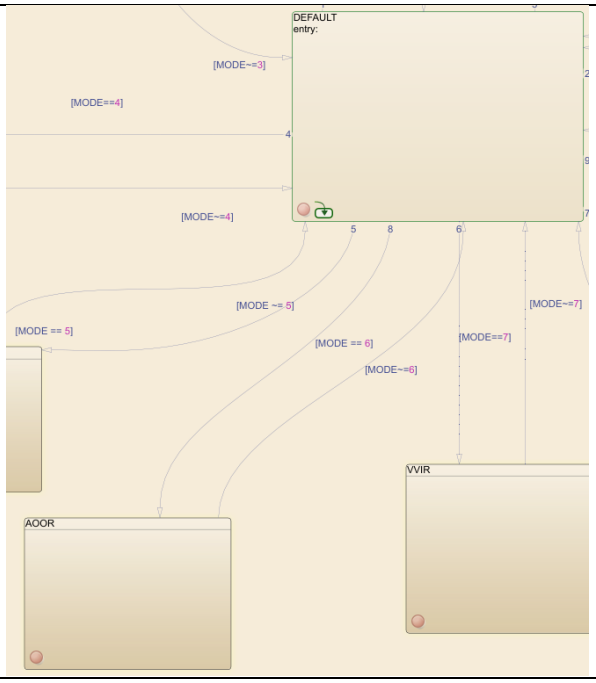


Fig. A

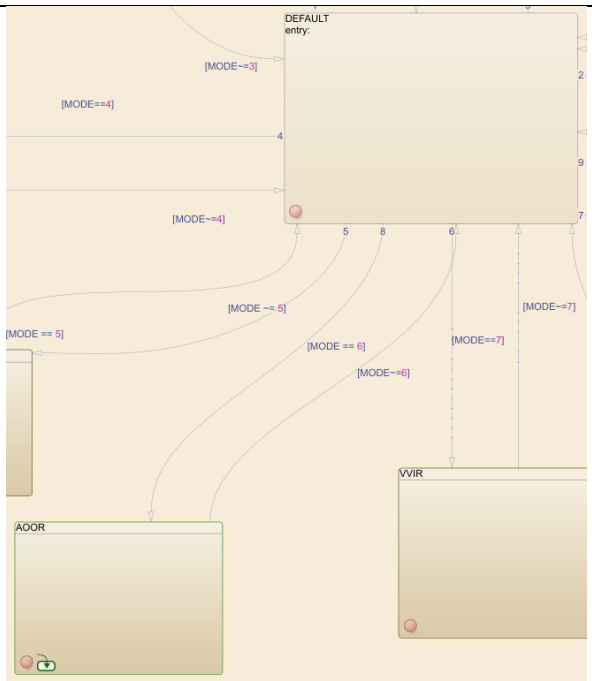


Fig. B

When the MODE = 6, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the AOOR state. Therefore, the mode change into AOOR with MODE = 6 works as intended.

MODE = 7

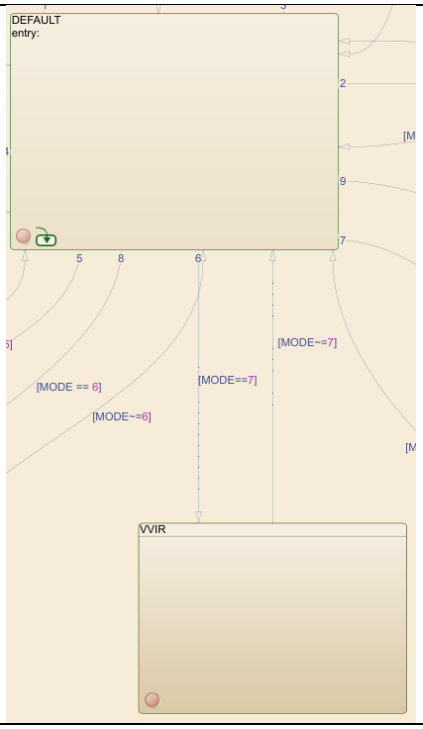


Fig. A

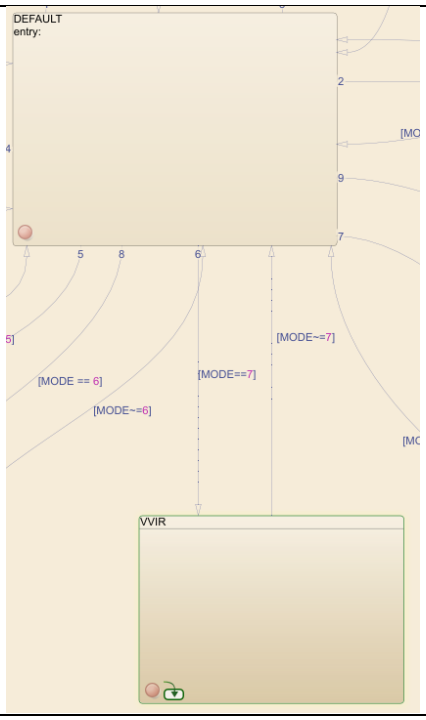


Fig. B

When the MODE = 7, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the VVIR state. Therefore, the mode change into VVIR with MODE = 7 works as intended.

MODE = 8

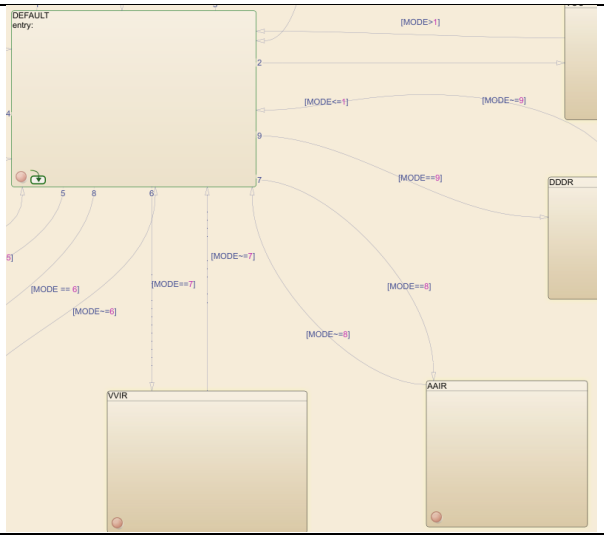


Fig. A

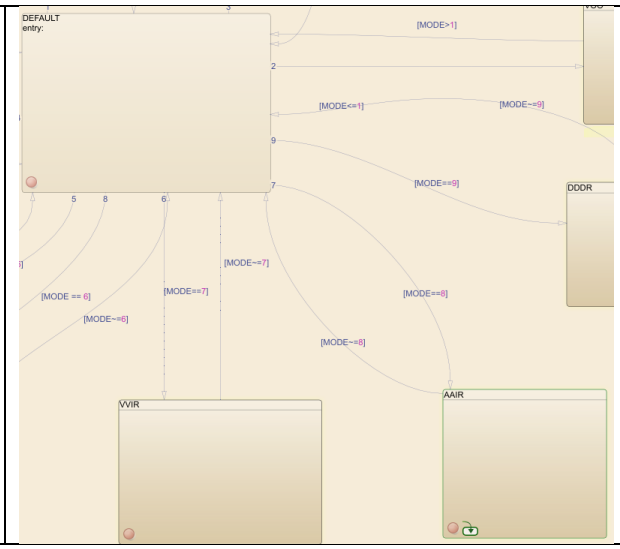


Fig. B

When the MODE = 8, in Fig. A, the DEFAULT state is highlighted in green signifying that the program is currently in that state. When continuing into the next step into Fig. B, the green highlighting transfers around the AAIR state. Therefore, the mode change into AAIR with MODE = 8 works as intended.

CurrRate = 80 ppm
LRL = 60 ppm
URL = 120 ppm
HRRL = 70 ppm

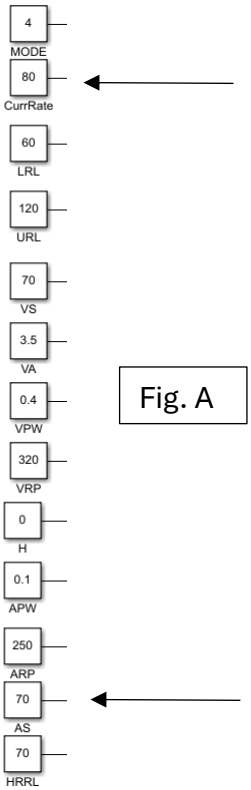


Fig. A

CurrRate	750	
CurRate	750	
FRONTEND_CTRL	0	Fig. B
hyst_wait	0	
HRRL	857.14	
HRL	857.14	

Both CurrRate and HRRL between intervals

Since the CurrRate is between LRL and URL, as well as HRRL is between LRL and CurrRate, there should be no change in either CurRate or HRL. This is depicted in as in Fig. A, the CurrRate and HRL are both set to 80 and 70 and after conversion they would be about 750 and 857.14 respectively. In Fig. B, looking at the Symbols Pane, CurRate is set to 750 and HRL is set to 857.14. Therefore, the CurrRate check in DEFAULT is working as intended.

CurrRate = 180 ppm
LRL = 60 ppm
URL = 120 ppm
HRRL = 70 ppm

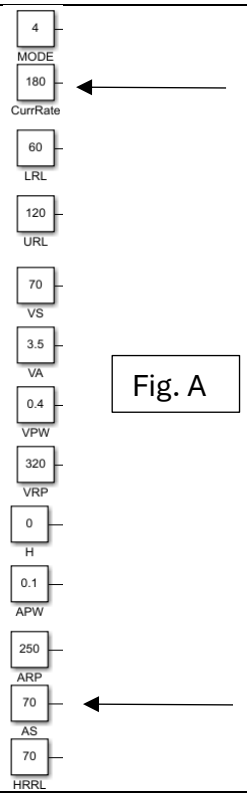


Fig. A

CurrRate	333.33	
CurRate	500	
FRONTEND_CTRL	0	Fig. B
hyst_wait	0	
HRRL	857.14	
HRL	857.14	

Test to show increased CurrRate past URL

Since the CurrRate is greater URL, while HRRL is between LRL and CurrRate, there should only be a change in CurRate. This is depicted in as in Fig. A, the CurrRate and HRL are both set to 180 and 70 and after conversion they would be about 333.33 and 857.14 respectively. In Fig. B, looking at the Symbols Pane, CurRate is set to 500 (defaulted to URL) and HRL is set to 857.14. Therefore, the CurrRate check in DEFAULT is working as intended.

CurrRate = 50 ppm
LRL = 60 ppm
URL = 120 ppm
HRRL = 60 ppm

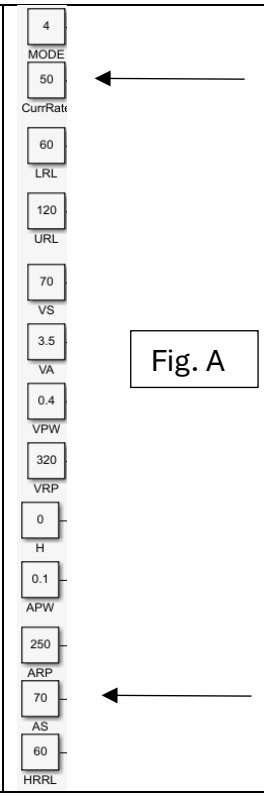


Fig. A

	CurrRate	1200	18	
	CurRate	1000		
	FRONTEND_CTRL	0	12	Fig. B
	hyst_wait	0		
	HRRL	1000	19	
	HRL	1000		

Test to show decreased CurrRate past LRL

Since the CurrRate is less than URL, while HRRL is between LRL and CurrRate, there should only be a change in CurRate. This is depicted in as in Fig. A, the CurrRate and HRL are both set to 50 and 60 and after conversion they would be about 1200 and 1000 respectively. In Fig. B, looking at the Symbols Pane, CurRate is set to 1000 (defaulted to LRL) and HRL is set to 1000. Therefore, the CurrRate check in DEFAULT is working as intended.

CurrRate = 80 ppm
LRL = 60 ppm
URL = 120 ppm
HRRL = 90 ppm

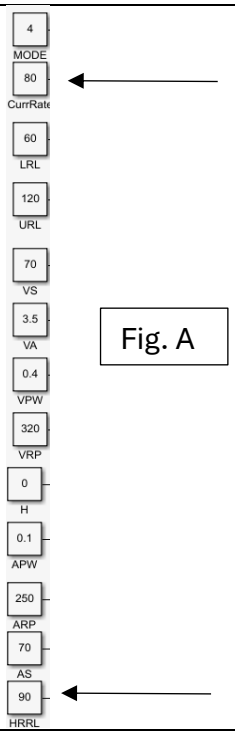


Fig. A

	CurrRate	750	18	
	CurRate	750		
	FRONTEND_CTRL	0	12	Fig. B
	hyst_wait	0		
	HRRL	666.66(	19	
	HRL	750		

Test to show increased HRRL past CurrRate

Since the HRRL is greater than CurrRate, while CurrRate is between LRL and URL, there should only be a change in HRL. This is depicted in as in Fig. A, the CurrRate and HRL are both set to 80 and 90 and after conversion they would be about 750 and 666.67 respectively. In Fig. B, looking at the Symbols Pane, CurRate is set to 750 and HRL is set to 750 (defaulted to CurrRate). Therefore, the HRRL check in DEFAULT is working as intended.

CurrRate = 80 ppm
LRL = 60 ppm
URL = 120 ppm
HRRL = 50 ppm

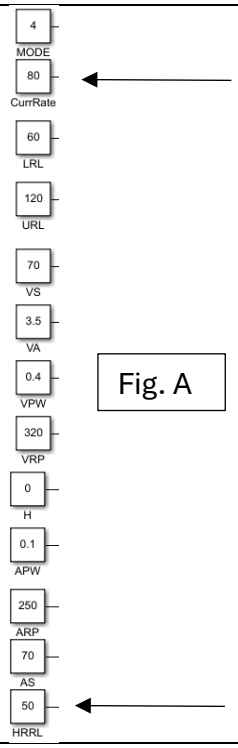


Fig. A

	CurrRate	750	18	
	CurRate	750		
	FRONTEND_CTRL	0	12	Fig. B
	hyst_wait	0		
	HRRL	1200	19	
	HRL	1000		

Test to show decreased HRRL past LRL

Since the HRRL is less than LRL, while CurrRate is between LRL and URL, there should only be a change in HRL. This is depicted in as in Fig. A, the CurrRate and HRL are both set to 80 and 50 and after conversion they would be about 750 and 1200 respectively. In Fig. B, looking at the Symbols Pane, CurRate is set to 750 and HRL is set to 1200 (defaulted to LRL). Therefore, the HRRL check in DEFAULT is working as intended.

5.13.2 Cohesion and Coupling

The Simulink model sought to produce a high-quality system design, particularly regarding modularity, maintainability, and code reuse, while keeping the concepts of cohesion and coupling in mind throughout the design process.

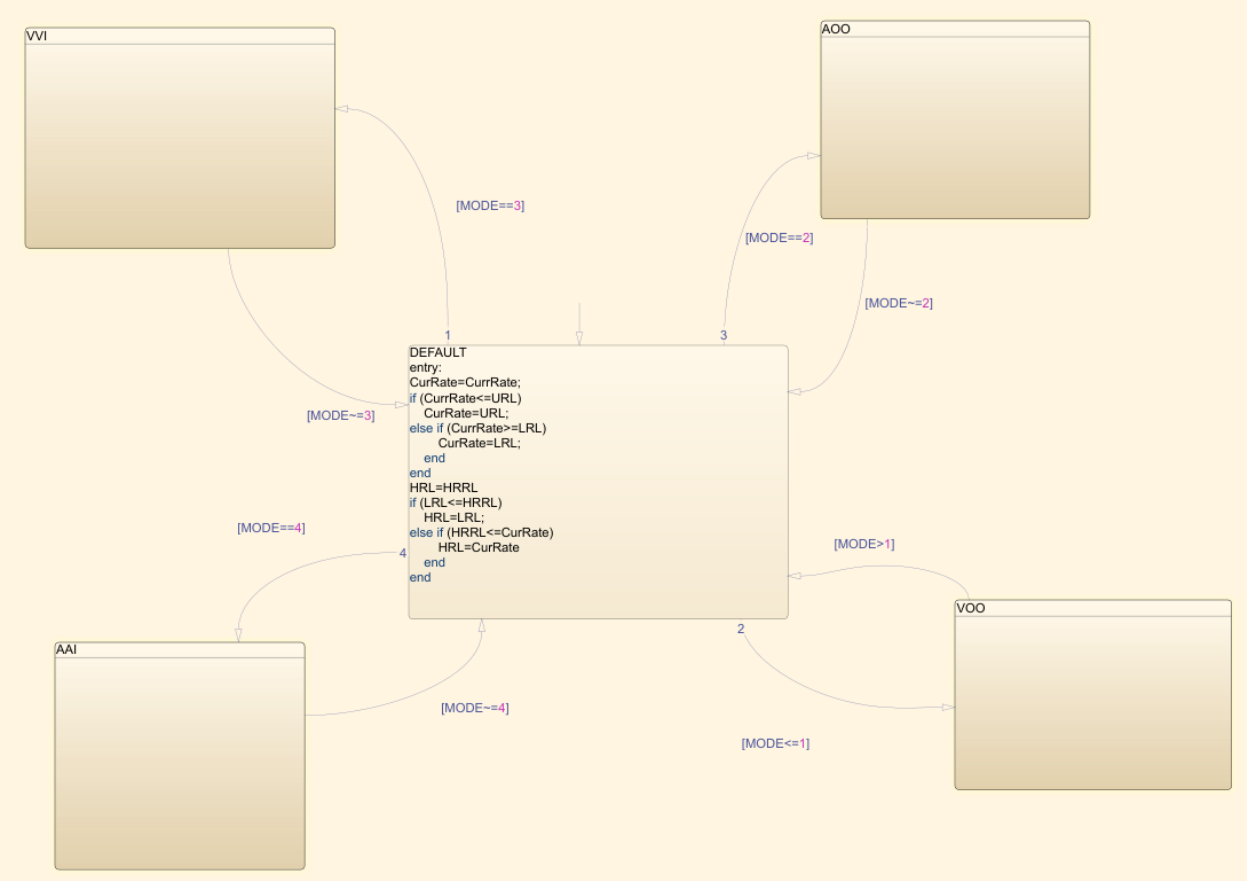


Figure 30: Simulink MODES Chart.

5.13.2.1 Cohesion

Each of the modes (AOO, VOO, AAI and VVI) is concentrated on a single, clearly defined function that sets it apart from the others, which is why the design indicates high cohesion. The independent blocks in the diagram (refer to Figure 9) represent the behaviour and responsibilities that each mode encapsulates. This hiding guarantees that each mode is cohesive since each mode is only focused on carrying out its own distinct task, without interfering with the duties of other modes. High cohesion is advantageous as it facilitates maintenance and clarifies the operation of the system.

5.13.2.2 Coupling

Since the modes only interact through the central "DEFAULT" state (refer to Figure 9), which regulates the transitions between them, the design exhibits low coupling. As control is centralised, there is less direct dependence between modes, preventing them from sharing or depending on one another's internal data or behaviour while maintaining flexibility.

5.13.3 Information Hiding

5.13.3.1 Hardware Hiding

October 25th, 2024.

The purpose of hardware hiding is to create a barrier between the software and the particulars of the hardware. Hardware hiding is used in the Simulink model at both the inputs and the outputs, where the pins that the PACEMAKER board is reading and writing to are concealed.

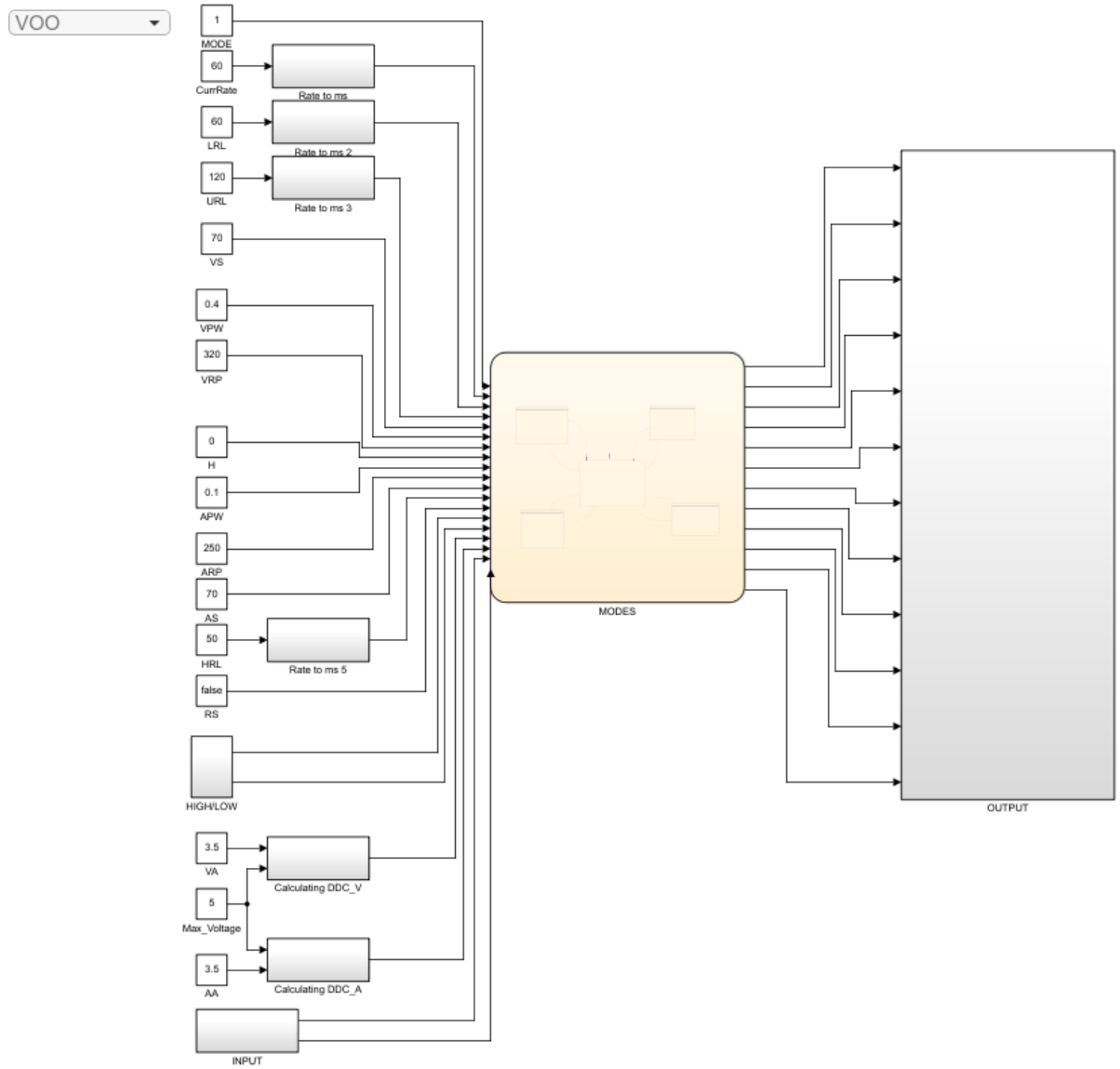


Figure 31: Simulink hardware hiding.

The Subsystem block that comes with Simulink is used in the hardware hiding implementation to handle inputs and outputs from the Subsystem blocks and to house all the read/write pins. The pins from the PACEMAKER board needed to sense and pace the heart in the specified modes are found in the Subsystems INPUT and OUTPUT. The pins would be provided after the Subsystems were inspected.

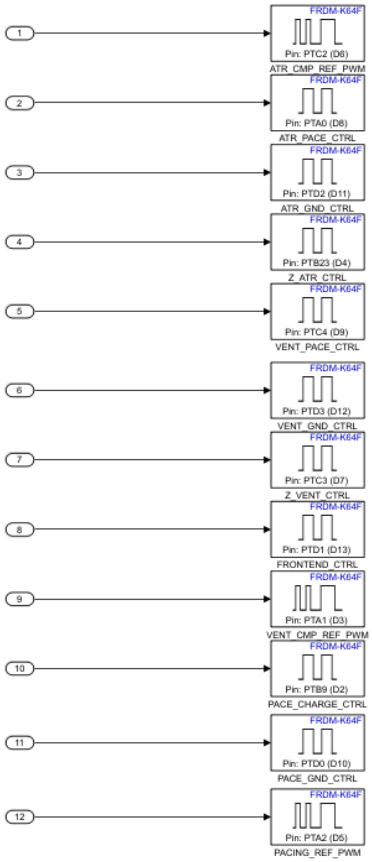


Figure 32: Simulink OUTPUT Subsystem for hardware hiding.

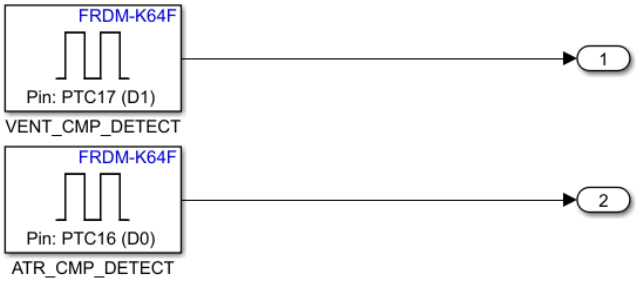


Figure 33: Simulink INPUT Subsystem for hardware hiding.

November 29th, 2024.

The new Simulink Model incorporates a more modular design that further separates all three of the different modules and uses “goto” and “from” blocks to allow each module to be an isolated system from the others. There is still hardware hiding such that all the pins to read and write are all still in their own separate subsystems, not directly connected to the main charts.

5.13.3.2 Behavior Hiding

October 25th, 2024.

Behaviour hiding isolates a component's internal operations from the rest of the system, much like hardware hiding does. Four constant inputs—CurrRate, LRL, URL, and HRL—have an additional Subsystem before they can be used as inputs into the MODES Chart, as shown in Figure 10. The "Rate to ms," "Rate to ms 2," "Rate to ms 3," and "Rate to ms 4" subsystems, which link these constants to the MODE Chart, all serve the same purpose of converting the heart rate from pulse per minute (ppm) to milliseconds per pulse. Using the following math, the conversion is carried out inside the Subsystem:

$$60\frac{s}{min} \times 1000\frac{ms}{s} \times \left(\frac{pulse}{min}\right)^{-1} = \frac{ms}{pulse}$$

$$60{,}000\frac{ms}{min} \div \frac{pulse}{min} = \frac{ms}{pulse}$$

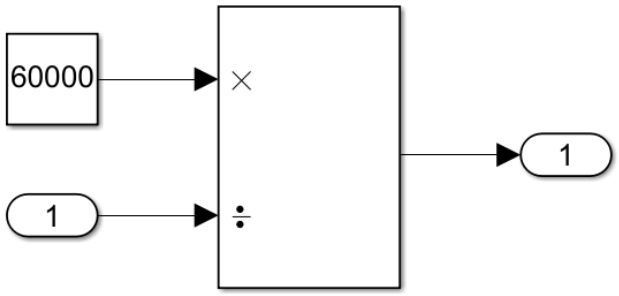


Figure 34: Simulink "Rate to ms" Subsystem for hardware hiding.

In addition to the four constant inputs listed above, there are three constants, VA, AA and Max_Voltage, which are used to calculate DDC_V and DDC_A values. There are two Subsystems ("Calculate DDC_V" and "Calculate DDC_A") that contain the same equations to convert the given constants into a DDC percentage that is used as an input for the MODES Chart. The calculations for both DDC_V and DDC_A are given below:

$$\frac{VA}{Max\_Voltage} \times 100\% = DDC\_V$$

$$\frac{AA}{Max\_Voltage} \times 100\% = DDC\_A$$

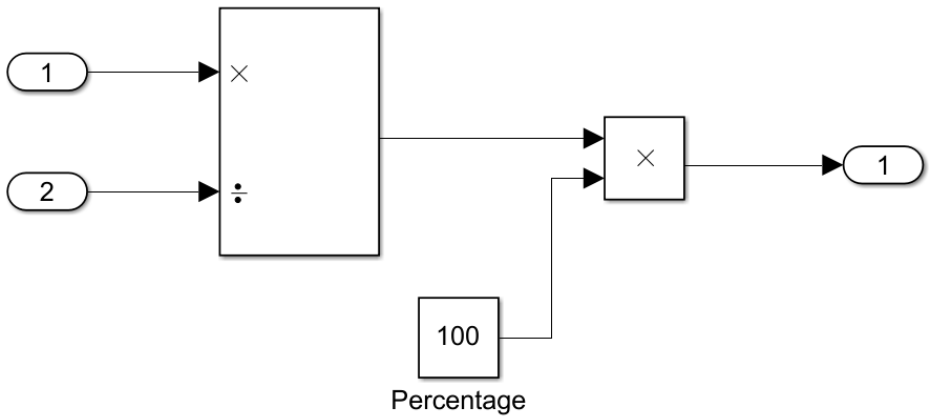


Figure 35: Simulink "Calculate DDC_V" Subsystem for hardware hiding.

November 29th, 2024.
The new model still includes behavior hiding such that the above subsystems are still embedded, along with new subsystems such as the subsystems in Rate Adaptive Pacing, refer to **5.5.3.6** and **5.5.3.7**.

5.14 Validation and Verification

Overall, this Simulink design underwent extensive validation and verification to make sure every part works as intended and satisfies the requirements. To ensure that it fulfils its intended functionality, each mode—VOO, AOO, VVI, and AAI—as well as the central DEFAULT state—was tested separately. Functional checks for each mode's primary duties and boundary conditions were part of the tests to ensure their dependability in a range of situations. However, slight deviations have occurred with both the VOO and AOO modes between the set DDC and pulse amplitude compared to the values illustrated in the charts. To guarantee accurate mode switching, the transitions between modes were also carefully studied. A thorough validation that the design is reliable, coherent, and appropriate for its intended use was provided by the documentation of all results, which captured the behaviour of each component in both expected and unlikely scenarios. This thorough validation and verification procedure grants that the design fulfils functional and operational requirements and operates as intended. Further testing in the future following updates to the modes, especially VOO and AOO, along with the introduction of new modes and requirement changes will be required.

Overall, this Simulink design underwent extensive validation and verification to make sure every part works as intended and satisfies the requirements. To ensure that it fulfills its intended functionality, each mode—VOO, AOO, VVI, AAI, VOOR, AOOR, VVIR and AAIR—as well as the central DEFAULT state—was tested separately. Functional checks for each mode's primary duties and boundary conditions were part of the tests to ensure their dependability in a range of situations. To guarantee accurate mode switching, the transitions between modes were also carefully studied. A thorough validation that the design is reliable, coherent, and appropriate for its intended use was provided by the

documentation of all results, which captured the behaviour of each component in both expected and unlikely scenarios. This thorough validation and verification procedure grants that the design fulfills functional and operational requirements and operates as intended.

6 DCM

6.1 Requirements

6.1.1 Login Page

The user must be able input a username and password to log into an existing account, which allows them to access the main interface.
Once logged in, the user must be taken to the main page.
A user must be able to access the registration page from the login page.
A user must be notified and denied access if their username and password are incorrect.
A user must be redirected to the login page if they are not logged in.

6.1.2 Registration Page

A user must be able to register a new account by inputting a name and password.
If there are already 10 existing accounts, a user attempting to register a new account will be denied and notified.
During registration, if the inputted username is already being used, the user will be denied and notified.
A user must input a password twice to confirm.
A user must be able to access the login page from the registration page.

6.1.3 Main Page

A user must be able to see their own username.
A user must be able to see the current date and time.
A user must be able to see when a pacemaker is connected.
A user must be able to log out from the Main Page.
A user must be able to see DCM information, specifically the pacemaker model number, the pacemaker software revision number, the DCM software revision number, and the institution name.
A user must be able to press a button to “interrogate” a pacemaker, which will redirect them to the parameter page.

6.1.4 Parameter Editing Page

A user must be able to see their own username.
A user must be able to see the current date and time.
A user must be able to see when a pacemaker is connected.
A user must be able to edit the appropriate pacemaker parameters (depends on current mode) while the pacemaker is connected to the DCM.
A user must be able to view the nine pacemaker modes (AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, VVIR, DDDR), as well as which one the pacemaker is currently on, and which one is currently being edited/viewed.
A user must be able to save the parameters to the DCM’s database.

If the pacemaker is disconnected, the user must be denied access to changing the parameters and the mode.
If no data is found in the database, the parameter values must be defaulted to the nominal values.
The parameter values must be restricted to the ranges and increments seen in the PACEMAKER document section A.
The user must be able to open a separate page showing the graphs page.
The user must be able to press a button to start sending requests for egram data.
The user must be able to press a button to stop sending requests for egram data.
The user must be denied permission to save parameters if the lower rate limit is set to a value higher than upper rate limit.
The user must be denied permission to save parameters if the maximum sensor rate (in modes where applicable) is set to a value outside of the bounds of lower and upper rate limit.
The user must be able to send parameters to the pacemaker over serial communication by pressing a button.
A user must be informed when the parameters echoed from the pacemaker do not match the sent parameters.

6.1.5 Graphs Page

A user must be able to see two graphs, one for atrium egram data and one for ventricle egram data.
A user must be able to close the graphs page by clicking a button.
Both the atrium and ventricle graph must hold at least 800 data points

6.2 Design Decisions

A light color scheme was chosen for the design of the DCM for the DCM to be more of a fit inside a sterile hospital environment. Additionally, a light theme is more accessible to users with astigmatism.

The DCM is developed using a framework called Reflex in the programming language Python. Reflex combines the power and flexibility of web development with a simplified development flow entirely in Python. This allowed us to also use the various libraries and utilities available in Python. Python was also a good choice because the team was already familiar with it.

User data is stored in a sqlite3 database. The choice of sqlite3 was to allow more flexibility while also having the stability and robustness of a well-tested database.

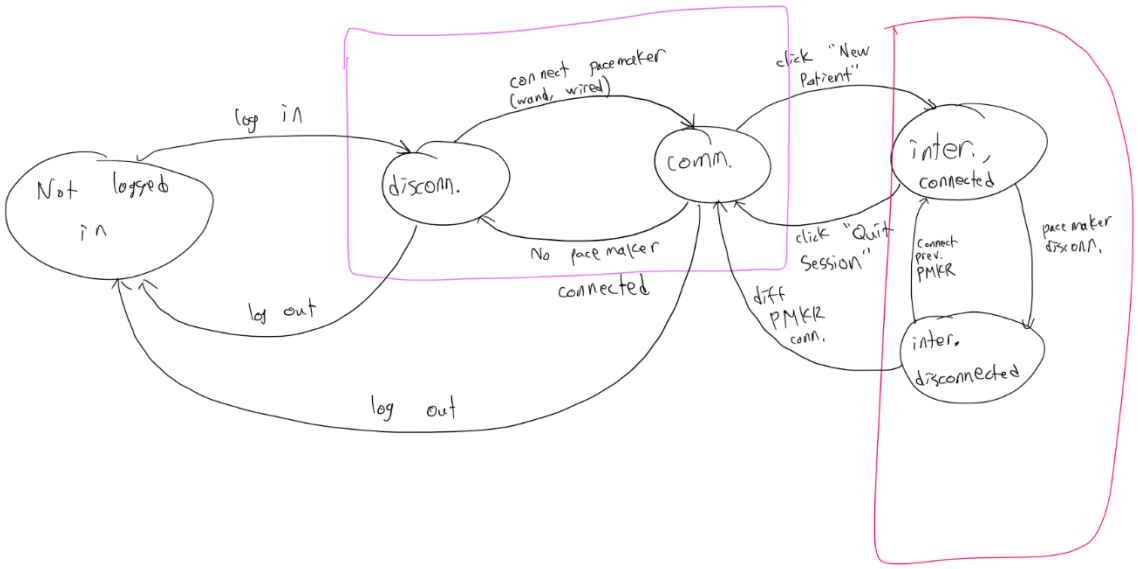
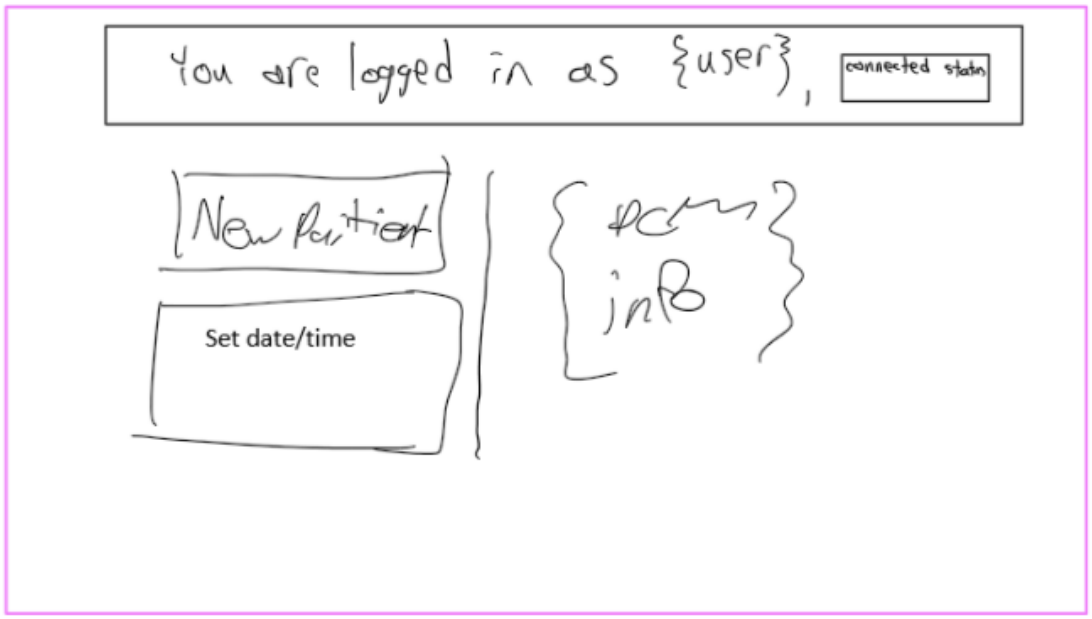


Figure 36: Initial State Diagram

The entire DCM is designed around the state diagram above with individual user visible pages boxed. Additionally, sketches of how certain pages in the DCM were created to help guide the development process by providing a goal and a base point that can be built towards or from as shown below.



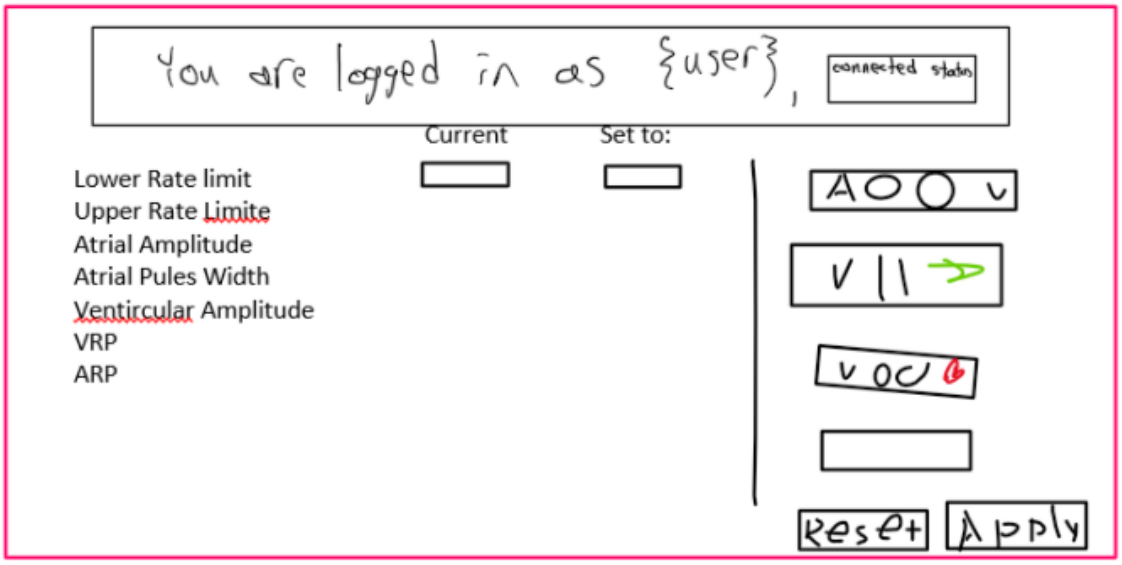


Figure 37: Initial sketches for UI

These design decisions evolved over the course of the development process and the DCM as of this assignment looks similar but not the same as our initial sketches.

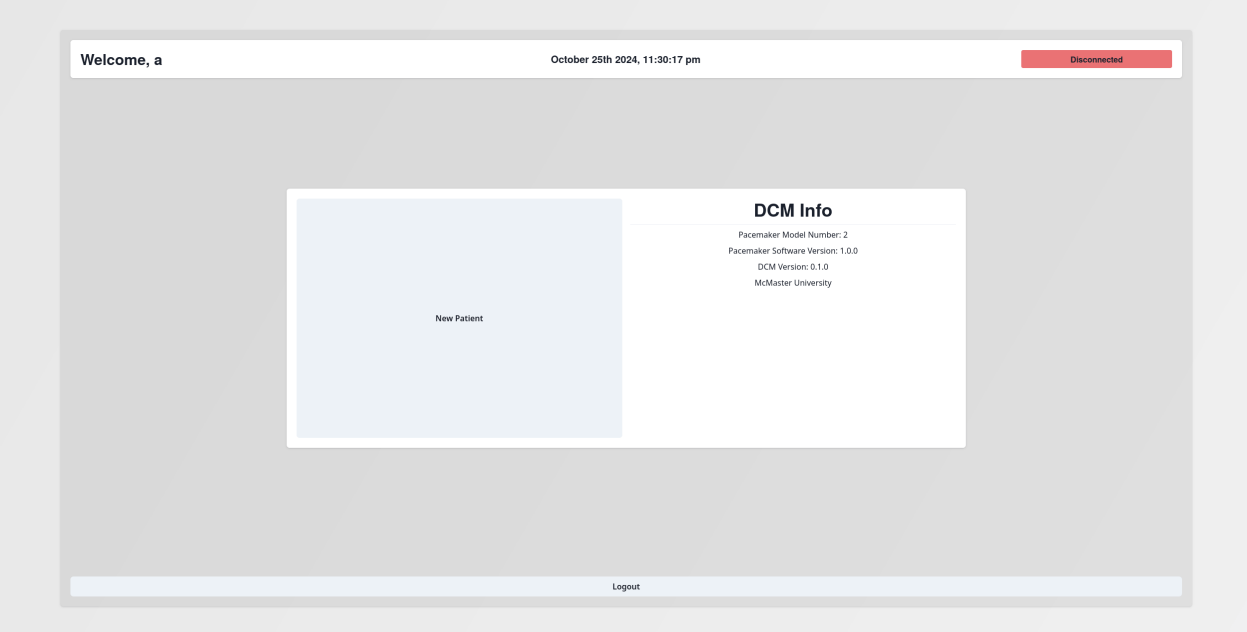


Figure 38: Initial version of main page

A key design choice that was made was to ensure that the DCM would be accessible using a touch interface, leading to large buttons and an overall simpler user interface. This serves two purposes; it makes the DCM more accessible and lowers the learning curve that the DCM would have.

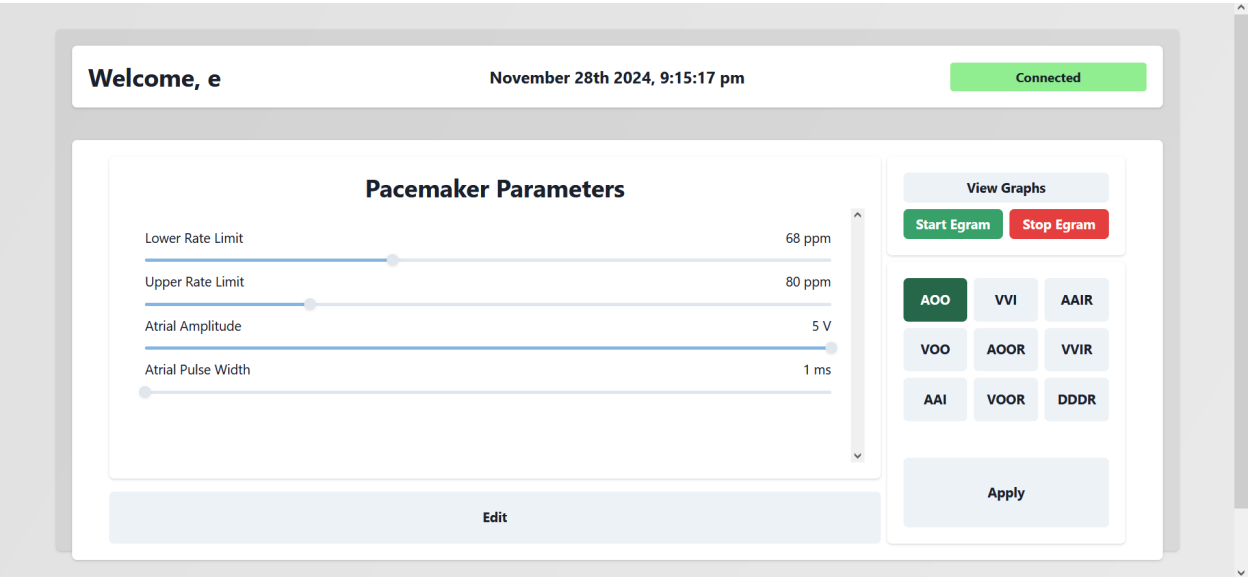


Figure 39: Final version of parameter editing page

Going into assignment 2, we continued using the design principles laid out during assignment 1. Touchscreen capability was maintained, and the overall UI design was not

changed. The main difference is the addition of new modes for the pacemaker, and a button to display egram graphs, as per assignment 2 requirements.

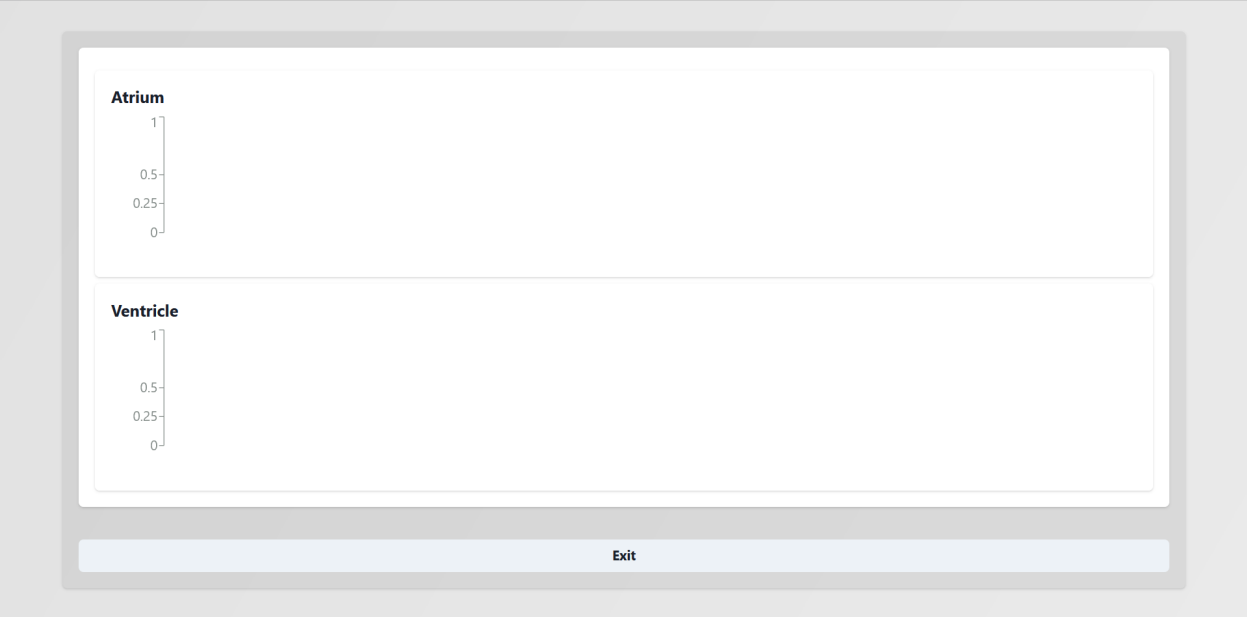


Figure 40: Final version of graphs page, no data shown

The graphs page was designed to be in a separate window from the main DCM. This would allow users to view both the graphs and the main page simultaneously. The main advantage of this is to allow users to edit and apply parameters while viewing the graphs, allowing them to see the changes immediately. Due to this, the graphs page is missing some of the elements shared by the other pages, such as the time and date, and the connected status of the device, as those are already displayed on the main page.

When sending messages over serial, we used a data format that ensured a constant message length. The message consisted of a sync byte, a function code, and a series of data bytes, followed by a checksum byte. The data bytes contain all parameters, even if the current mode does not use them. This method sends unnecessary data, but allows the pacemaker side to simply ignore the data bytes assigned to the parameters that are unused, making development significantly easier. This format will be discussed further in the Serial Communications section.

The DCM also includes parameter safety checks before sending to pacemaker as part of redundancy, diversity, and defense-in-depth. The pacemaker requests an echo of parameters as soon as it sends parameters. It then checks against that echoed set to ensure that the parameters were set correctly. If not, the user is notified. This is another added layer of protection on top of the checksum check that occurs on both the pacemaker and DCM.

6.3 Modules

The DCM is mainly split into 3 overarching components. These are the entrypoint, backend, and the frontend. The split is also motivated by the way Reflex is designed to work. Additionally, due to the constraints that Reflex applies on the way the program must be written, the DCM does not utilize classes in the usual sense. Additional helper files that don't make up a whole module or are shared between multiple modules are not listed.

The entrypoint is the where the DCM starts from and is responsible for setting up Reflex and launching a webview to ensure a native app experience. The entrypoint will also be responsible for handling communication with the pacemaker in the future.

The backend holds all the state that the frontend might need to display the correct information to the user. This state includes stuff such as the currently logged in user, and values and limits of input sliders. The state is explained in more detail below where we treat each state class as a module due to the design choices of Reflex making it so that we do not use classes in the usual sense.

6.3.1 dcm.py

This module is the entrypoint into the DCM. This module handles launching the Reflex application frontend and backend servers as well as launching a webview to present the website as a native application. In assignment 2, background tasks were added to facilitate the sending and receiving of messages over serial.

```python
# register app
app = rx.App(
  theme=rx.theme(appearance="light", has_background=True, radius="large", accent_color="mint", panel_background="translucent", gray_color="auto")
)
app.add_page(login, route="/login")
app.add_page(register, route="/register")
app.add_page(index, route="/", on_load=State.on_load())
app.add_page(parameters, route="/parameters", on_load=State.on_load())
app.add_page(graph, route="/graph")

# task to handle communication with a simulated or real pacemaker
SERIAL_PORTS = ["/dev/ttyUSB0", "/dev/ttyACM0", "COM3", "COM4", "COM5"]
async def communication_task():
  while True:
    if BackendState().connected_device is None:
      for port in SERIAL_PORTS:
        logging.debug(f"trying to connect to {port}")
        try:
          reader, writer = await serial_asyncio.open_serial_connection(url=port, baudrate=115200)
          logging.info("connected to serial device")
          BackendState().connected_device = (reader, writer)

          # if egram was running before put something into the queue
          if BackendState().egram_running:
            await BackendState().egram_queue.coro_put(1)

          break
        except serial.serialutil.SerialException:
          BackendState().connected_device = None

    # check if device is still connected by trying to read from it
    if BackendState().connected_device is not None:
      reader, writer = BackendState().connected_device
      try:
        data = await reader.read(0)
      except serial.serialutil.SerialException:
        logging.error("connection to device lost")
        BackendState().connected_device = None
```

dcm > dcm.py > ...

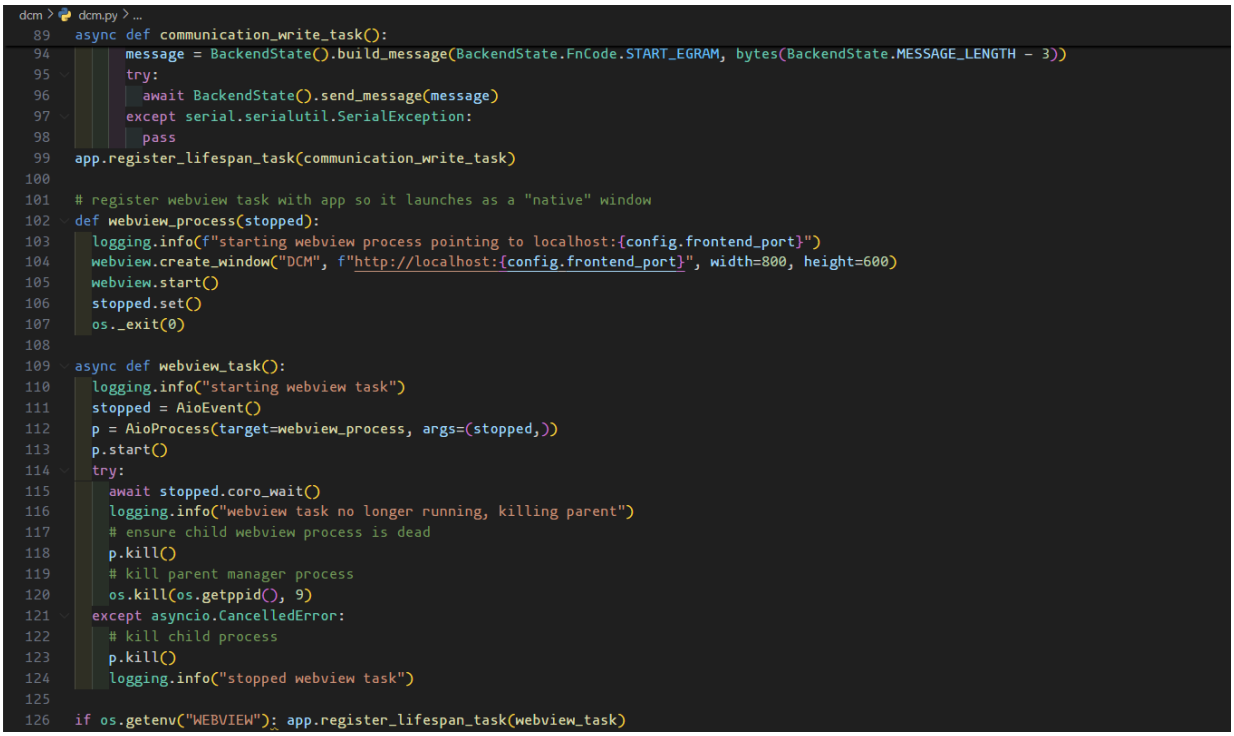
```python
async def communication_task():
    await asyncio.sleep(0.1)
app.register_lifespan_task(communication_task)
async def communication_read_task():
  while True:
    try:
      msg = await BackendState().read_message()
    except serial.serialutil.SerialException:
      msg = None
    if msg:
      logging.info(f"received message: {msg.hex()}, {msg[1]}")
      if msg[1] != BackendState.FnCode.EGRAM_DATA.value and msg[1] != BackendState.FnCode.START_EGRAM.value:
        await BackendState().msg_queue.coro_put(msg)
      else:
        # parse egram data message, sent at the end of the message as 2 doubles
        atrium, ventricle = struct.unpack("dd", msg[-17:-1])
        print(atrium, ventricle)

        # remove old data point
        if len(BackendState().atrium_egram) > BackendState.EGRAM_POINTS:
          _ = BackendState().atrium_egram.pop(0)
          _ = BackendState().ventricle_egram.pop(0)

        # add new data point
        t = time.monotonic_ns()
        BackendState().atrium_egram.append({"data": atrium, "time": t})
        BackendState().ventricle_egram.append({"data": ventricle, "time": t})

        # queue a new message to update the graph
        if BackendState().egram_running:
          await BackendState().egram_queue.coro_put(1)

    await asyncio.sleep(0.01)
app.register_lifespan_task(communication_read_task)
```

dcm > dcm.py > ...

```python
# register app
app = rx.App(
  theme=rx.theme(appearance="light", has_background=True, radius="large", accent_color="mint", panel_background="translucent", gray_color="auto")
)
app.add_page(login, route="/login")
app.add_page(register, route="/register")
app.add_page(index, route="/", on_load=State.on_load())
app.add_page(parameters, route="/parameters", on_load=State.on_load())
app.add_page(graph, route="/graph")

# task to handle communication with a simulated or real pacemaker
SERIAL_PORTS = ["/dev/ttyUSB0", "/dev/ttyACM0", "COM3", "COM4", "COM5"]
async def communication_task():
  while True:
    if BackendState().connected_device is None:
      for port in SERIAL_PORTS:
        logging.debug(f"trying to connect to {port}")
        try:
          reader, writer = await serial_asyncio.open_serial_connection(url=port, baudrate=115200)
          logging.info("connected to serial device")
          BackendState().connected_device = (reader, writer)

          # if egram was running before put something into the queue
          if BackendState().egram_running:
            await BackendState().egram_queue.coro_put(1)

          break
        except serial.serialutil.SerialException:
          BackendState().connected_device = None

    # check if device is still connected by trying to read from it
    if BackendState().connected_device is not None:
      reader, writer = BackendState().connected_device
      try:
        data = await reader.read(0)
      except serial.serialutil.SerialException:
        logging.error("connection to device lost")
        BackendState().connected_device = None
```

dcm > dcm.py > ...

```python
async def communication_read_task():
  while True:
    try:
      msg = await BackendState().read_message()
    except serial.serialutil.SerialException:
      msg = None
    if msg:
      logging.info(f"received message: {msg.hex()}, {msg[1]}")
      if msg[1] != BackendState.FnCode.EGRAM_DATA.value and msg[1] != BackendState.FnCode.START_EGRAM.value:
        await BackendState().msg_queue.coro_put(msg)
      else:
        # parse egram data message, sent at the end of the message as 2 doubles
        atrium, ventricle = struct.unpack("dd", msg[-17:-1])
        print(atrium, ventricle)

        # remove old data point
        if len(BackendState().atrium_egram) > BackendState.EGRAM_POINTS:
          _ = BackendState().atrium_egram.pop(0)
          _ = BackendState().ventricle_egram.pop(0)

        # add new data point
        t = time.monotonic_ns()
        BackendState().atrium_egram.append({"data": atrium, "time": t})
        BackendState().ventricle_egram.append({"data": ventricle, "time": t})

        # queue a new message to update the graph
        if BackendState().egram_running:
          await BackendState().egram_queue.coro_put(1)

    await asyncio.sleep(0.01)
app.register_lifespan_task(communication_read_task)
async def communication_write_task():
  while True:
    await BackendState().egram_queue.coro_get()

    if BackendState().connected_device is not None and BackendState().egram_running:
      message = BackendState().build_message(BackendState.FnCode.START_EGRAM, bytes(BackendState.MESSAGE_LENGTH - 3))
      try:
        await BackendState().send_message(message)
```



```
dcm > dcm.py > ...
async def communication_write_task():
        message = BackendState().build_message(BackendState.FnCode.START_EGRAM, bytes(BackendState.MESSAGE_LENGTH - 3))
        try:
            await BackendState().send_message(message)
        except serial.serialutil.SerialException:
            pass
app.register_lifespan_task(communication_write_task)

# register webview task with app so it launches as a "native" window
def webview_process(stopped):
    logging.info(f"starting webview process pointing to localhost:{config.frontend_port}")
    webview.create_window("DCM", f"http://localhost:{config.frontend_port}", width=800, height=600)
    webview.start()
    stopped.set()
    os._exit(0)

async def webview_task():
    logging.info("starting webview task")
    stopped = AioEvent()
    p = AioProcess(target=webview_process, args=(stopped,))
    p.start()
    try:
        await stopped.coro_wait()
        logging.info("webview task no longer running, killing parent")
        # ensure child webview process is dead
        p.kill()
        # kill parent manager process
        os.kill(os.getppid(), 9)
    except asyncio.CancelledError:
        # kill child process
        p.kill()
        logging.info("stopped webview task")

if os.getenv("WEBVIEW"): app.register_lifespan_task(webview_task)
```

Figure 41: A collection of code screenshots showing the variables and functions of the dcm.py module

Table 47: Global Variables

Variable	Type	Description
app	rx.App	The Reflex application container for the entire DCM.

Table 48: dcm.py Functions

Function	Arguments	Return	Description
communication_task	None	None	Lifespan task that checks connection with pacemaker
webview_process	stopped	None	Child process function that launches and monitors the webview instance.
webview_task	None	None	Lifespan task that manages the child webview process and propagates quit requests the Reflex manager process.
communication_read_task	None	None	Lifespan task that handles reading data coming from the pacemaker

communication_write_task		None	Lifespan task that handles sending egram requests to pacemaker

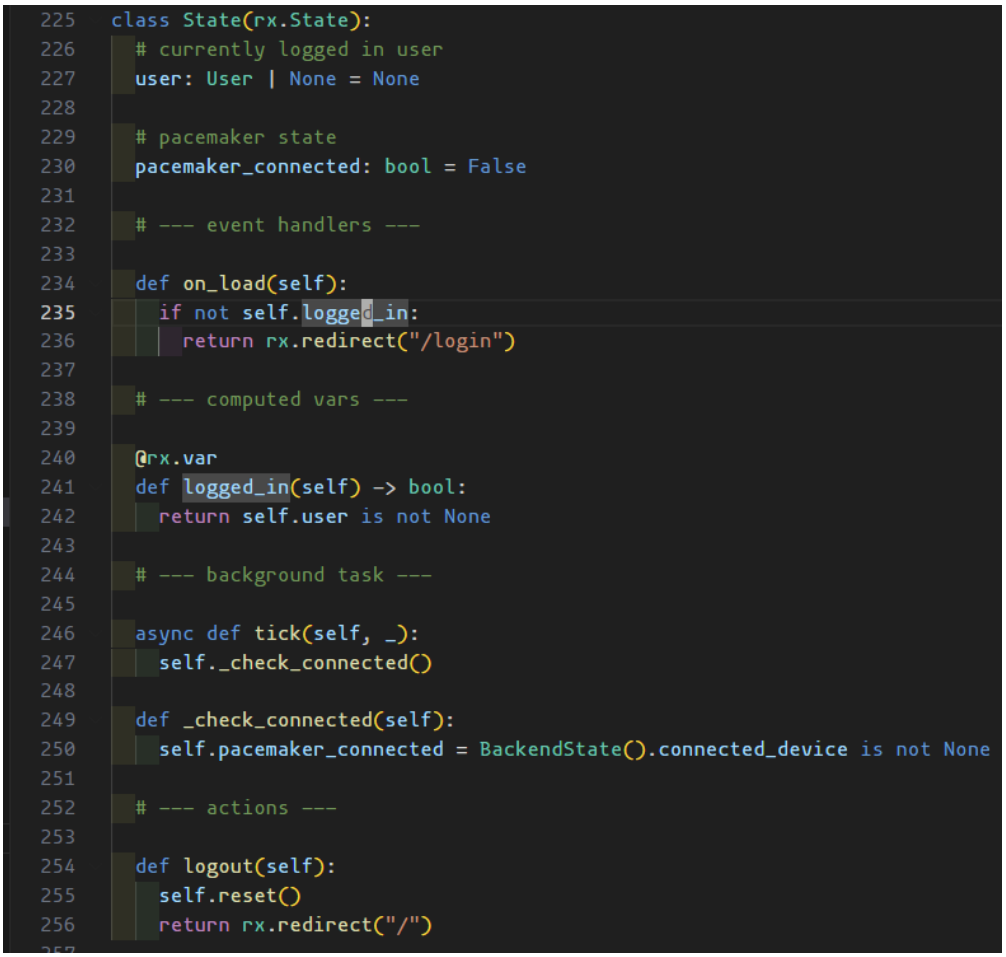
6.3.2 state.py

This module contains and computes all the backend state for the user facing frontend. This module is further split into several submodules that inherit from a single parent module to allow common state to be shared while having separation on page specific state.

During assignment 2, state.py was expanded to have more parameters and modes, as well as the ability to send messages over serial.

6.3.2.1 State

This is the base and common state for everything. It handles the management of the long-lived background task that monitors changes from the connected pacemaker as well as running checks to ensure that a user is logged in before access to a certain page is allowed.



```
225  class State(rx.State):
226      # currently logged in user
227      user: User | None = None
228
229      # pacemaker state
230      pacemaker_connected: bool = False
231
232      # --- event handlers ---
233
234      def on_load(self):
235          if not self.logged_in:
236              return rx.redirect("/login")
237
238      # --- computed vars ---
239
240      @rx.var
241      def logged_in(self) -> bool:
242          return self.user is not None
243
244      # --- background task ---
245
246      async def tick(self, _):
247          self._check_connected()
248
249      def _check_connected(self):
250          self.pacemaker_connected = BackendState().connected_device is not None
251
252      # --- actions ---
253
254      def logout(self):
255          self.reset()
256          return rx.redirect("/")
```

Figure 42: State class code screenshot

Table 49: State Variables

Variable	Type	Description
user	User	Stores the currently logged in user.
_ticking	bool	Stores if the background task is running or not in order to ensure that only a single background task is running.
now	datetime.datetime	Stores the current time to display in the header.
pacemaker_connnected	bool	Stores the status of the pacemaker connection. True if the pacemaker is connected and False if the pacemaker is not connected.

Table 50: State Functions

Function	Arguments	Return	Description
on_load	self	None	Event handler that is called whenever a page is loaded, stops any existing background task. Then checks if the user is logged in or not, if they are not logged in, it redirects them to the login page. If they are logged in, it starts the background task.
logged_in	self	bool	A computed variable/property that returns if the current user is logged in or not by checking if self.user is None.
tick	self	None	The asynchronous background task function. Checks connectivity with dcm.py about the pacemaker connection and updates the clock every second.
_refresh_now	self	None	Private helper method that only updates self.now every second to ensure that the number of state updates is kept low. Removed in assignment 2

_check_connected	self	None	Private helper method that checks with the BackendState singleton on if there is a connected pacemaker.
logout	self	None	Resets the state and logs the user out.

6.3.2.2 BackendState

This state is a singleton that exists across all possible frontend sessions that holds an exclusive lock on the connected pacemaker. This state is here to ensure that if multiple sessions are in communication with the same pacemaker all configuration requests and writes happen with a deterministic order and are not racing against each other. All backend states interface with this state to communicate with the connected pacemaker. In assignment 2, BackendState handles serial communication with the pacemaker.

```python
class BackendState(metaclass=Singleton):
    DDDR_ENABLED = False
    MESSAGE_LENGTH = 54 if DDDR_ENABLED else 43 #used to be 27
    EGRAM_POINTS = 800

    def __init__(self):
        self.connected_device: tuple[asyncio.StreamReader, asyncio.StreamWriter] | None = None
        self.msg_queue: AioQueue = AioQueue()

        self.egram_running: bool = False
        self.egram_queue: AioQueue = AioQueue()
        self.atrium_egram: list[dict[str, float]] = []
        self.ventricle_egram: list[dict[str, float]] = []

    class FnCode(Enum):
        START_EGRAM = 0x47
        STOP_EGRAM = 0x62
        SEND_PARAMS = 0x55
        RECV_PARAMS = 0x49

        EGRAM_DATA = 0x88

    def f_chk(self, data: bytes) -> bytes:
        return bytes([reduce(lambda x, y: x ^ y, data)])

    def build_message(self, fn: FnCode, data: bytes) -> bytes:
        assert len(data) == BackendState.MESSAGE_LENGTH - 3, f"data must be {BackendState.MESSAGE_LENGTH - 3} bytes long"
        return bytes([0x16, fn.value]) + data + self.f_chk(data)

    def build_params_message(self, user_set: ParameterSet) -> bytes:
        match user_set.mode:
            case "VOO": data = bytes([0x01])
            case "AOO": data = bytes([0x02])
            case "VVI": data = bytes([0x03])
            case "AAI": data = bytes([0x04])
            case "VOOR": data = bytes([0x05])
            case "AOOR": data = bytes([0x06])
            case "VVIR": data = bytes([0x07])
```

Figure 43: BackendState code screenshot

Table 51: BackendState Variables

Variable	Type	Description

connected_device	tuple[asyncio.StreamReader, asyncio.StreamWriter]	Replaces pacemaker_connected. Uses the serial connection to check if a device is connected.
msg_queue	AioQueue	Queue of egram messages to send to pacemaker
egram_running	bool	Boolean that determines if egram requests are currently being sent
atrium_egram	list[dict[str, float]]	Shift buffer of data points to be displayed on atrium egram graph
ventricle_egram	list[dict[str, float]]	Shift buffer of data points to be displayed on ventricle egram graph

Table 52: BackendState Functions

Function	**Arguments**	**Return**	**Description**
interrogate	self	None	Loads device's current mode and sends the user to the parameter editing page
quit_interrogate	self	None	Returns user to home page
start_egram	self	None	Sends egram request messages to pacemaker
stop_egram	self	None	Stops sending of egram requests to pacemaker
save	self	None	Saves current parameters to database, performs safety checks on LRL, URL, and MSR to ensure LRL is not higher than URL, and MSR is within LRL/URL bounds.
edit	self	None	Puts app into editing mode, allowing the user to edit parameters
apply_pending	self	None	Sends current parameters and mode to pacemaker to be applied. Once sent, sends a request for the pacemaker to send parameters back. Performs a safety check to ensure that pacemaker

			parameters match the sent parameters.
build_params_message	self, user_set	bytes	Builds the data section of the parameter message to be sent to the pacemaker, returns it in bytes
build_message	self, fncode, data	bytes	Build the entire message to be sent to the pacemaker, returns it in bytes
f_chk	self, data	bytes	Generates a 1 byte checksum based on the data passed in
send_message	self, message	None	Send message passed in over serial to pacemaker
read_message	self,	bytes	Reads message and returns it as bytes

6.3.2.3 AuthState

This state handles registration and user log in. It restricts the number of registrations to 10, and checks usernames and passwords upon log in to check if they match an existing user. It also checks usernames with existing usernames to ensure no duplicate users are created by registration. There are no variables defined inside of this state.

```python
class AuthState(State):
  def register(self, form_data: dict[str, str]):
    username = form_data.get("username")
    if not username:
      return rx.window_alert("Username is required")
    password = form_data.get("password")
    if not password:
      return rx.window_alert("Password is required")
    password2 = form_data.get("password2")
    if password != password2:
      return rx.window_alert("Passwords do not match")

    with rx.session() as session:
      if len(session.exec(select(User)).all()) == 10:
        return rx.window_alert("10 users exist already")
      if session.exec(select(User).where(User.username == username)).first():
        return rx.window_alert("User already exists")

      user = User(username=username, password=password)
      session.add(user)
      session.expire_on_commit = False
      session.commit()

    self.reset()
    return rx.redirect("/login")

  def login(self, form_data: dict):
    username = form_data.get("username")
    if not username:
      return rx.window_alert("Username is required")
    password = form_data.get("password")
    if not password:
      return rx.window_alert("Password is required")

    with rx.session() as session:
      user = session.exec(select(User).where(User.username == username)).first()
      if user and user.password == password:
        self.user = user
        return rx.redirect("/")
      else:
        return rx.window_alert("Invalid username or password")
```

Figure 44: Code screenshot of AuthState

Table 53: AuthState Functions

Function	Arguments	Return	Description
register	form_data	rx.window_alert rx.redirect	Attempts to register a user, using username and password from form_data, and returns an alert if 10 users already exist, or if a username is already taken. Otherwise, it returns a redirect object to take the user back to the login page.
login	form_data	rx.window_alert rx.redirect	Attempts to log in using the username and password from form_data. Returns a redirect object to the main page if successful, and a window alert otherwise.

6.3.2.4 ParametersState

This state handles the entire configuration page for the pacemaker. It enforces limits on the possible configuration values and handles saving and loading configuration values from the database. It also handles when a parameter is able to be edited and seen, depending on the current mode.

```
258  class ParametersState(State):
259      current_mode: rx.Field[str] = rx.field("AOO")
260
261      pending_lower_rate_limit: rx.Field[int] = rx.field(0)
262      pending_upper_rate_limit: rx.Field[int] = rx.field(0)
263      pending_atrial_amplitude: rx.Field[int] = rx.field(0) # divided by 10
264      pending_atrial_pulse_width: rx.Field[int] = rx.field(0)
265      pending_atrial_refractory_period: rx.Field[int] = rx.field(0)
266      pending_ventricular_amplitude: rx.Field[int] = rx.field(0) # divided by 10
267      pending_ventricular_pulse_width: rx.Field[int] = rx.field(0)
268      pending_ventrical_refractory_period: rx.Field[int] = rx.field(0)
269      pending_maximum_sensor_rate: rx.Field[int] = rx.field(0)
270      pending_post_ventricular_atrial_refractory_period: rx.Field[int] = rx.field(0)
271      pending_hysteresis: rx.Field[int] = rx.field(0)
272      pending_rate_smoothing: rx.Field[int] = rx.field(0)
273      pending_activity_threshold: rx.Field[int] = rx.field(0)
274      pending_reaction_time: rx.Field[int] = rx.field(0)
275      pending_response_factor: rx.Field[int] = rx.field(0)
276      pending_recovery_time: rx.Field[int] = rx.field(0)
277      pending_atrial_sensitivity: rx.Field[int] = rx.field(0)
278      pending_ventricular_sensitivity: rx.Field[int] = rx.field(0)
279      #
280      pending_fixed_av_delay: rx.Field[int] = rx.field(0)
281      pending_dynamic_av_delay: rx.Field[int] = rx.field(0)
282      pending_min_dynamic_av_delay: rx.Field[int] = rx.field(0)
283      pending_sensed_av_delay_offset: rx.Field[int] = rx.field(0)
284      pending_pvarp_extension: rx.Field[int] = rx.field(0)
285      pending_atr_duration: rx.Field[int] = rx.field(0)
286      pending_atr_fallback_mode: rx.Field[int] = rx.field(0)
287      pending_atr_fallback_time: rx.Field[int] = rx.field(0)
288
289      pending_mode: rx.Field[str] = rx.field("AOO")
290
291      editing: bool = False
292
293      @rx.var(cache=True)
294      def pending_lower_rate_limit_shown(self):
295          return True
```

Figure 45: ParametersState code screenshot, shows state variables

```
300     def pending_atrial_amplitude_shown(self):
301         return self.pending_mode in ("AOO", "AAI", "AOOR", "AAIR", "DDDR")
302     @rx.var(cache=True)
303     def pending_atrial_pulse_width_shown(self):
304         return self.pending_mode in ("AOO", "AAI", "AOOR", "AAIR", "DDDR")
305     @rx.var(cache=True)
306     def pending_atrial_refractory_period_shown(self):
307         return self.pending_mode in ("AAI", "AAIR", "DDDR")
308     @rx.var(cache=True)
309     def pending_ventricular_amplitude_shown(self):
310         return self.pending_mode in ("VOO", "VVI", "VOOR", "VVIR", "DDDR")
311     @rx.var(cache=True)
312     def pending_ventricular_pulse_width_shown(self):
313         return self.pending_mode in ("VOO", "VVI", "VOOR", "VVIR", "DDDR")
314     @rx.var(cache=True)
315     def pending_ventrical_refractory_period_shown(self):
316         return self.pending_mode in ("VVI", "VVIR", "DDDR")
317     @rx.var(cache=True)
318     def pending_maximum_sensor_rate_shown(self):
319         return self.pending_mode in ("AOOR", "AAIR", "VOOR", "VVIR", "DDDR")
320     @rx.var(cache=True)
321     def pending_post_ventricular_atrial_refractory_period_shown(self):
322         return self.pending_mode in ("AAIR", "AAI", "DDDR")
323     @rx.var(cache=True)
324     def pending_hysteresis_shown(self):
325         return self.pending_mode in ("AAIR", "AAI", "VVI", "VVIR", "DDDR")
326     @rx.var(cache=True)
327     def pending_rate_smoothing_shown(self):
328         return self.pending_mode in ("AAIR", "AAI", "VVI", "VVIR", "DDDR")
329     @rx.var(cache=True)
330     def pending_activity_threshold_shown(self):
331         return self.pending_mode in ("AAIR", "VVIR", "VOOR", "AOOR", "DDDR")
332     @rx.var(cache=True)
333     def pending_reaction_time_shown(self):
334         return self.pending_mode in ("AAIR", "VVIR", "VOOR", "AOOR", "DDDR")
```

Figure 46: ParametersState code screenshot, shows the selection for which parameters are shown, depending on mode

```python
@rx.var(cache=True)
def pending_lower_rate_limit_translated(self) -> int:
    if(self.pending_lower_rate_limit <= 4):
        return 30 + self.pending_lower_rate_limit * 5
    elif(self.pending_lower_rate_limit <= 44):
        return 50 + (self.pending_lower_rate_limit - 4)
    else:
        return 90 + (self.pending_lower_rate_limit - 44) * 5

def pending_lower_rate_limit_untranslate(self, data: int) -> int:
    if(30 <= data <= 50):
        return int((data-30)/5)
    elif(51 <= data <= 90):
        return int((data-51) + 5)
    else:
        return int((data-90)/5 + 44)

@rx.var(cache=True)
def pending_atr_duration_translated(self) -> int:
    if(self.pending_atr_duration == 0):
        return 10
    elif(self.pending_atr_duration <= 5):
        return 20 * self.pending_atr_duration
    else:
        return 100 * (self.pending_atr_duration-4)

def pending_atr_duration_untranslate(self, data) -> int:
    if(data <= 10):
        return 0
    elif(data <= 80):
        return int(data/20)
    else:
        return int((data/100) + 4)
```

Figure 47: ParametersState code screenshot, shows slider translation functions

```
with rx.session() as session:
    user_set: ParameterSet | None = session.exec(ParameterSet.select().where(ParameterSet.user == self.user).where(ParameterSet.mode == self.pending_mode)).first()
    if user_set:
        self.pending_lower_rate_limit = int(self.pending_lower_rate_limit_untranslate(user_set.lower_rate_limit))
        self.pending_upper_rate_limit = int(user_set.upper_rate_limit)
        self.pending_atrial_amplitude = int(user_set.atrial_amplitude * 10)
        self.pending_atrial_pulse_width = int(user_set.atrial_pulse_width)
        self.pending_atrial_refractory_period = int(user_set.atrial_refractory_period)
        self.pending_ventricular_amplitude = int(user_set.ventricular_amplitude * 10)
        self.pending_ventricular_pulse_width = int(user_set.ventricular_pulse_width)
        self.pending_ventrical_refractory_period = int(user_set.ventrical_refractory_period)
        self.pending_maximum_sensor_rate = int(user_set.maximum_sensor_rate)
        self.pending_post_ventricular_atrial_refractory_period = int(user_set.post_ventricular_atrial_refractory_period)
        self.pending_hysteresis = int(self.pending_hysteresis_untranslate(user_set.hysteresis))
        self.pending_rate_smoothing = int(user_set.rate_smoothing)
        self.pending_activity_threshold = int(user_set.activity_threshold)
        self.pending_reaction_time = int(user_set.reaction_time)
        self.pending_response_factor = int(user_set.response_factor)
        self.pending_recovery_time = int(user_set.recovery_time)
        self.pending_atrial_sensitivity = int(user_set.atrial_sensitivity * 10)
        self.pending_ventricular_sensitivity = int(user_set.ventricular_sensitivity * 10)
        self.pending_fixed_av_delay = int(user_set.fixed_av_delay)
        self.pending_dynamic_av_delay = int(user_set.dynamic_av_delay)
        self.pending_min_dynamic_av_delay = int(user_set.min_dynamic_av_delay)
        self.pending_sensed_av_delay_offset = int(user_set.sensed_av_delay_offset)
        self.pending_pvarp_extension = int(user_set.pvarp_extension)
        self.pending_atr_duration = int(self.pending_atr_duration_untranslate(user_set.atr_duration))
        self.pending_atr_fallback_mode = int(user_set.atr_fallback_mode)
        self.pending_atr_fallback_time = int(user_set.atr_fallback_time)
    else:
```

Figure 48: ParametersState *code screenshot, shows the storing of parameters in the database*

```
async def apply_pending(self):
    self.save()
    self.current_mode = self.pending_mode

    with rx.session() as session:
        user_set: ParameterSet | None = session.exec(ParameterSet.select().where(ParameterSet.user == self.user).where(ParameterSet.mode == self.pending_mode)).first()

        if not user_set:
            return rx.window_alert("No parameters set for this mode")

        message = BackendState().build_params_message(user_set)
        await BackendState().send_message(message)

        # check if the data was written to the device
        message = BackendState().build_message(BackendState.FnCode.RECV_PARAMS, bytes(BackendState.MESSAGE_LENGTH - 3))
        await BackendState().send_message(message)
        # read the response
        response = await BackendState().msg_queue.coro_get()
        res_set = BackendState().parse_params_message(response)

        # check if parameters pacemaker sends back are the same
        if(BackendState.DDDR_ENABLED):
            if (
                user_set.lower_rate_limit != res_set.lower_rate_limit or
                user_set.upper_rate_limit != res_set.upper_rate_limit or
                user_set.atrial_amplitude != res_set.atrial_amplitude or
                user_set.atrial_pulse_width != res_set.atrial_pulse_width or
                user_set.atrial_refractory_period != res_set.atrial_refractory_period or
                user_set.ventricular_amplitude != res_set.ventricular_amplitude or
                user_set.ventricular_pulse_width != res_set.ventricular_pulse_width or
                user_set.ventrical_refractory_period != res_set.ventrical_refractory_period or
                user_set.maximum_sensor_rate != res_set.maximum_sensor_rate or
                user_set.post_ventricular_atrial_refractory_period != res_set.post_ventricular_atrial_refractory_period or
                user_set.hysteresis != res_set.hysteresis or
                user_set.rate_smoothing != res_set.rate_smoothing or
```

Figure 49: ParametersState code screenshot, shows the sending of parameters and the proceeding check

```python
async def interrogate(self):
    if not self.pacemaker_connected:
        return rx.window_alert("No pacemaker connected")

    message = BackendState().build_message(BackendState.FnCode.RECV_PARAMS, bytes(BackendState.MESSAGE_LENGTH - 3))
    await BackendState().send_message(message)
    response = await BackendState().msg_queue.coro_get()
    res_set = BackendState().parse_params_message(response)
    self.current_mode = res_set.mode

    self.set_pending_mode(self.current_mode)

    return rx.redirect("/parameters")

def quit_interrogate(self):
    self.reset()
    return rx.redirect("/")

def view_graphs(self):
    return rx.call_script(f"window.open('{self.router.page.host}/graph', '_blank', 'width=800,height=600')")
    # return rx.redirect("/graph", external=True)
```

Figure 50: ParametersState code screenshot, shows the initial connection check

Table 54: ParametersState Variables

Variable	Type	Description
pending_lower_rate_limit	rx.Field[int]	Stores the current lower rate limit as set by the user
pending_upper_rate_limit	rx.Field[int]	Stores the current upper rate limit as set by the user
pending_atrial_amplitude	rx.Field[int]	Stores the atrial amplitude as set by the user
pending_atrial_pulse_width	rx.Field[int]	Stores the atrial pulse width as set by the user
pending_atrial_refractory_period	rx.Field[int]	Stores the atrial refractory period as set by the user
pending_ventricular_amplitude	rx.Field[int]	Stores the ventricular amplitude as set by the user
pending_ventricular_pulse_width	rx.Field[int]	Stores the ventricular pulse width as set by the user
pending_ventrical_refractory_period	rx.Field[int]	Stores the ventricular refractory period as set by the user
pending_mode	rx.Field[int]	Stores the current mode being viewed/edited by the user
editing	bool	Flag that indicates if a user is currently editing parameters
pending_maximum_sensor_rate	rx.Field[int]	Stores the maximum sensor rate being viewed/edited by the user
pending_post_ventricular_atrial_refractory_period	rx.Field[int]	Stores the PVARP value being viewed/edited by the user

pending_hysteresis	rx.Field[int]	Stores the hystersis value being viewed/edited by the user
pending_rate_smoothing	rx.Field[int]	Stores the rate smoothing value being viewed/edited by the user
pending_activity_threshold	rx.Field[int]	Stores the activity threshold value being viewed/edited by the user
pending_reaction_time	rx.Field[int]	Stores the reaction time value being viewed/edited by the user
pending_response_time	rx.Field[int]	Stores the response time value being viewed/edited by the user
pending_recovery_time	rx.Field[int]	Stores the recovery time value being viewed/edited by the user
pending_atrial_sensitivity	rx.Field[int]	Stores the atrial sensitivity value being viewed/edited by the user
pending_ventricular_sensitivity	rx.Field[int]	Stores the ventricular sensitivity value being viewed/edited by the user
pending_fixed_av_delay	rx.Field[int]	Stores the fixed av delay value being viewed/edited by the user
pending_dynamic_av_delay	rx.Field[int]	Stores the dynamic av delay boolean being viewed/edited by the user
pending_min_dynamic_av_delay	rx.Field[int]	Stores the minimum dynamic av delay value being viewed/edited by the user
pending_sensed_av_delay_offset	rx.Field[int]	Stores the sensed av delay offset value being viewed/edited by the user
pending_pvarp_extension	rx.Field[int]	Stores the PVARP extension value being viewed/edited by the user
pending_atr_duration	rx.Field[int]	Stores the ATR duration value being viewed/edited by the user
pending_atr_fallback_mode	rx.Field[int]	Stores the ATR fallback mode value being viewed/edited by the user
pending_atr_fallback_time	rx.Field[int]	Stores the ATR fallback time value being viewed/edited by the user

Table 55: ParametersState Functions

Function	**Arguments**	**Return**	**Description**
pending_lower_rate_limit_shown	self	bool	Returns a Boolean value indicating if lower rate limit is to be shown, depending on the current mode.

pending_upper_rate_limit_shown	self	bool	Returns a Boolean value indicating if upper rate limit is to be shown, depending on the current mode.
pending_atrial_amplitude_shown	self	bool	Returns a Boolean value indicating if atrial amplitude is to be shown, depending on the current mode.
pending_atrial_pulse_width_shown	self	bool	Returns a Boolean value indicating if atrial pulse width is to be shown, depending on the current mode.
pending_atrial_refractory_period_shown	self	bool	Returns a Boolean value indicating if atrial refractory period is to be shown, depending on the current mode.
pending_ventricular_amplitude_shown	self	bool	Returns a Boolean value indicating if ventricular amplitude period is to be shown, depending on the current mode.
pending_ventricular_pulse_width_shown	self	bool	Returns a Boolean value indicating if ventricular pulse width is to be shown, depending on the current mode.
pending_ventrical_refractory_period_shown	self	bool	Returns a Boolean value indicating if ventricular refractory period is to be shown, depending on the current mode.
pending_lower_rate_limit_step	self	int	Returns an integer indicating the increment of the lower rate limit, depending on it's current value (lower ate limit has more granular increments in the middle of its range)
pending_atrial_pulse_width_display	self	str	Returns the value to be displayed on screen for the current value of the atrial pulse width
pending_ventricular_pulse_width_display	self	str	Returns the value to be displayed on screen for the current value of the ventricular pulse width
aoo_current	self	str	Returns "green" if the current mode is AOO, otherwise it returns "gray"
voo_current	self	str	Returns "green" if the current mode is VOO, otherwise it returns "gray"
aai_current	self	str	Returns "green" if the current mode is AAI, otherwise it returns "gray"
vvi_current	self	str	Returns "green" if the current mode is VVI, otherwise it returns "gray"
aoo_pending	self	bool	Returns True if the pending mode is AOO else returns False

voo_pending	self	bool	Returns True if the pending mode is VOO else returns False
aai_pending	self	bool	Returns True if the pending mode is AAI else returns False
vvi_pending	self	bool	Returns True if the pending mode is VVI else returns False
set_pending_mode	self	mode	Loads parameters from database, uses default nominal values if none are found.
apply_pending	self	none	Sets the current mode of the pacemaker to the mode being viewed by the user. In assignment 2, sends parameter message to pacemaker and requests parameters from pacemaker. It then checks parameters from the pacemaker against the ones that were just sent to ensure they were applied correctly.
pending_maximum_sensor_rate_shown	self	bool	Returns a Boolean value indicating if maximum sensor rate is to be shown, depending on the current mode.
pending_post_ventricular_atrial_refractory_period_shown	self	bool	Returns a Boolean value indicating if PVARP is to be shown, depending on the current mode.
pending_hysteresis_shown	self	bool	Returns a Boolean value indicating if hysteresis is to be shown, depending on the current mode.
pending_rate_smoothing_shown	self	bool	Returns a Boolean value indicating if rate smoothing is to be shown, depending on the current mode.
pending_activity_threshold_shown	self	bool	Returns a Boolean value indicating if the activity threshold is to be shown, depending on the current mode.
pending_reaction_time_shown	self	bool	Returns a Boolean value indicating if reaction time is to be shown, depending on the current mode.
pending_response_factor_shown	self	bool	Returns a Boolean value indicating if response factor is to be shown, depending on the current mode.
pending_recovery_time_shown	self	bool	Returns a Boolean value indicating if recovery time is to be shown, depending on the current mode.

pending_atrial_sensitivity_shown	self	bool	Returns a Boolean value indicating if atrial sensitivity is to be shown, depending on the current mode.
pending_ventricular_sensitivity_shown	self	bool	Returns a Boolean value indicating if ventricular sensitivity is to be shown, depending on the current mode.
pending_fixed_av_delay_shown	self	bool	Returns a Boolean value indicating if fixed av delay is to be shown, depending on the current mode.
pending_dynamic_av_delay_shown	self	bool	Returns a Boolean value indicating if dynamic av delay is to be shown, depending on the current mode.
pending_min_dynamic_av_delay_shown	self	bool	Returns a Boolean value indicating if min dynamic av delay is to be shown, depending on the current mode.
pending_sensed_av_delay_offset_shown	self	bool	Returns a Boolean value indicating if sensed av delay offset is to be shown, depending on the current mode.
pending_pvarp_extension_shown	self	bool	Returns a Boolean value indicating if PVARP extension is to be shown, depending on the current mode.
pending_atr_duration_shown	self	bool	Returns a Boolean value indicating if ATR duration is to be shown, depending on the current mode.
pending_atr_fallback_mode_shown	self	bool	Returns a Boolean value indicating if ATR fallback mode is to be shown, depending on the current mode.
pending_atr_fallback_time_shown	self	bool	Returns a Boolean value indicating if ATR fallback time is to be shown, depending on the current mode.
aoor_current	self	bool	Returns “green” if the current mode is AOOR, otherwise it returns “gray”
voor_current	self	bool	Returns “green” if the current mode is VOOR, otherwise it returns “gray”
aair_current	self	bool	Returns “green” if the current mode is AAIR, otherwise it returns “gray”
vvir_current	self	bool	Returns “green” if the current mode is VVIR, otherwise it returns “gray”
dddr_current	self	bool	Returns “green” if the current mode is DDDR, otherwise it returns “gray”
aoor_pending	self	bool	Returns True if the pending mode is AOOR else returns False

voor_pending	self	bool	Returns True if the pending mode is VOOR else returns False
aair_pending	self	bool	Returns True if the pending mode is AAIR else returns False
vvir_pending	self	bool	Returns True if the pending mode is VVIR else returns False
dddr_pending	self	bool	Returns True if the pending mode is DDDR else returns False
pending_lower_rate_limit_translated	self	int	Translates the value on the slider to a value within LRL bounds (needed due to dynamic increment on slider)
pending_lower_rate_limit_untranslate	self, data	int	Maps the values of LRL to the slider in the parameter page
pending_atr_duration_translated	self	int	Translates the value on the slider to a value within ATR duration bounds (needed due to dynamic increment on slider)
pending_atr_duration_untranslate	self, data	int	Maps the values of ATR duration to the slider in the parameter page
pending_hysteresis_translated	self	int	Translates the value on the slider to a value within hysteresis bounds (needed due to dynamic increment on slider)
pending_hysteresis_untranslate	self, data	int	Maps the values of hysteresis to the slider in the parameter page
pending_rate_smoothing_display	self	str	Returns the string used to label the rate smoothing slider in the UI
pending_rate_smoothing_stored	self	int	Translates the value on the slider to a value within rate smoothing bounds (needed due to dynamic increment on slider)
pending_activity_threshold_display	self	str	Returns the string used to label the activity threshold slider in the UI
pending_dynamic_av_delay_display	self	str	Returns the string used to label the dynamic av delay slider in the UI

pending_atr_fallback_mode_display	self	str	Returns the string used to label ATR fallback mode slider in the UI
pending_sensed_av_delay_offset_display	self	str	Returns the string used to label AV delay offset slider in the UI

6.3.2.5 GraphState

This state only holds data to be displayed on the egram graphs. It uses a shift buffer to hold this data, allowing old data to be discarded while refreshing with new data.

Table 56: GraphState Variables

Variable	Type	Description
atrium	list[dict[str, float]]	Stores the atrium egram data to be displayed on the graphs page
ventricle	list[dict[str, float]]	Stores the ventricle egram data to be displayed on the graphs page

Table 57: GraphState Functions

Function	Arguments	Return	Description
_update_data	self	None	Updates egram data with data in BackendState.
tick	self	None	Calls _update_data. Tick is used to set this as a background task that runs constantly.

6.4 Likely Changes

Assignment 1

Going into assignment 2, multiple requirements are likely to change or be added. A new page will likely need to be added to facilitate the display of graphs for egram data. As well, the ability to change to more modes than the current four will likely need to be added. Serial communication will also be required for assignment 2.

However, changes to our design decisions will likely be minimal. Assignment 1 was designed with the eventuality of assignment 2 in mind, so overarching decisions such as UI and credential storage are unlikely to change. Minor changes will need to be made to accommodate storage of the new parameters introduced in assignment 2.

Assignment 2

If the project were to be continued past assignment 2, some likely requirement changes would be to add the other pacemaker modes and parameters. Additionally, data for the pacing activity, rather than just the egram could be displayed on graphs.

There are multiple out of scope changes that could be made as well. Data encryption could be implemented for user credentials to improve security. Log files could be added to track changes made by users. A feature could be added to export pacemaker data to an external file.

6.5 Validation and Verification

The DCM was assessed according to the fundamental principles of design:

- DCM maps to the client's conceptual model using signifiers and affordances
    - Ex. Sliders for the programable parameters signifies to the user to slide it, and the user can slide it
- DCM incorporates extensive visible feedback
    - Ex. Green and red boxes for connected and disconnected status.

Table 58: Test cases

Justification	Input	Expected Output	Actual Output	Result
Login Page				
Standard use case for DCM	User attempts login with correct credentials	User is redirected to main page	As expected	Pass
Safety check	User attempts login with incorrect credentials	User is denied access and asked to log in again	As expected	Pass
Registration Page				
Standard use case	User registers a new account (unique username) while <10 users in database	Account is registered, user redirected to login page	As expected	Pass
Maximum capacity test	User registers a new account while 10 users in database	User is notified and denied permission to create account	As expected	Pass
Safety check	User registers an account with a username that already exists	User is notified and denied permission to create account	As expected	Pass
Main Page				
Standard use case	User presses "New Patient" while a device is connected	User is redirected to parameter editing page	As expected	Pass
Safety check	User presses "New Patient" while a device is not connected	User is notified and remains on main page	As expected	Pass
Standard use case	User presses "Logout"	User is redirected to login page and is logged out	As expected	Pass

Parameter Editing Page				
Standard use case	User presses “Edit” while pacemaker is connected	User is allowed to edit parameters	As expected	Pass
Safety check	User presses “Edit while pacemaker is not connected	User is denied permission to edit parameters, “Edit” button is grayed out	As expected	Pass
Standard use case	User is in editing mode, attempts to edit parameters while pacemaker is connected	Parameters are edited accordingly, sliders can be moved	As expected	Pass
Standard use case	User is in editing mode, attempts to edit parameters while pacemaker is disconnected	User cannot change parameters, sliders are unable to be moved	As expected	Pass
Standard use case	User is in editing mode, presses “Save” (assuming parameters are correct)	User exits from editing mode, parameters are saved to database	As expected	Pass
Safety check	User is in editing mode, presses “Save” while parameters are incorrect (LRL > URL, or MSR outside of LRL/URL bounds)	User is notified of error, parameters are not saved. User does not exit from editing mode.	As expected	Pass
Standard use case	User presses a button for a different mode than is currently being viewed, while editing mode is off	Page switches view to the mode selected	As expected	Pass
Safety check	User presses a button for a different mode than is currently being viewed, while editing mode is on	Page does not switch view, buttons for other modes are grayed out	As expected	Pass
Standard use case	User presses “Apply” while pacemaker is connected	Page sends current parameters over serial to pacemaker, mode being applied has its	As expected	Pass

		corresponding button turn green		
Safety check	User presses “Apply” while pacemaker is disconnected	No parameters are sent, apply button is grayed out, button does not change color	As expected	Pass
Standard use case	User presses “View Graphs”	Graphs page is opened	As expected	Pass
Safety check	User presses “View Graphs” multiple times	Multiple graphs pages are opened, all display same data	As expected	Pass
Standard use case	User presses “Start Egram”	egram data starts streaming and is viewable on graph on graphs page. If already streaming, does nothing	As expected	Pass
Standard use case	User presses “Stop Egram”	egram data stops streaming, graph is still viewable on graphs page. If egram data is not streaming, does nothing	As expected	Pass
Graphs Page				
Standard use case	User presses “Exit”	Graphs pages close	As expected	Pass

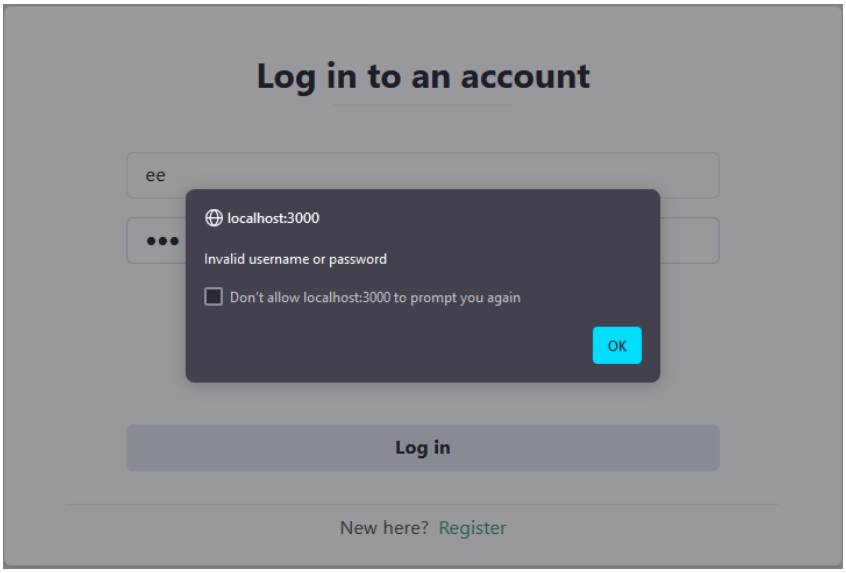


Figure 51: Result of incorrect credentials test

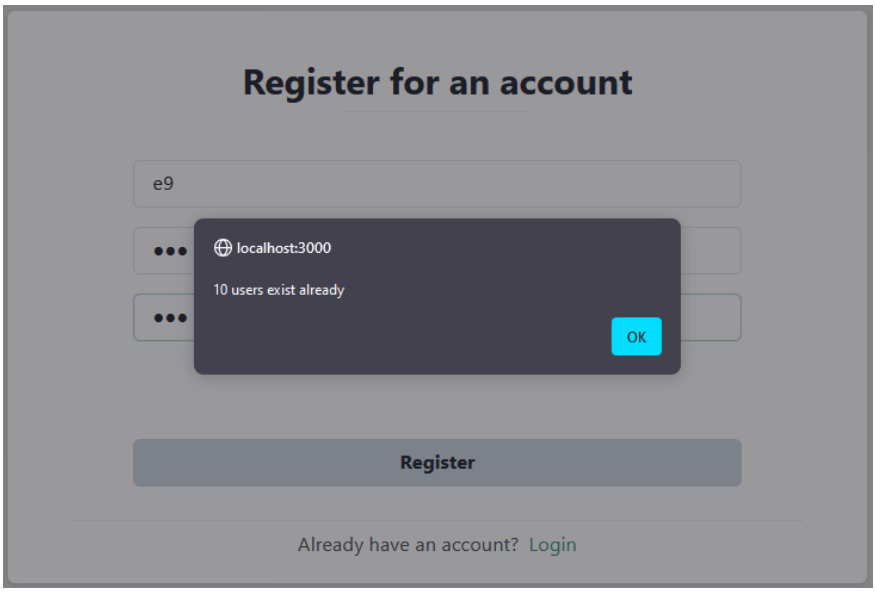


Figure 52: Result of attempting to register account while 10 users exist

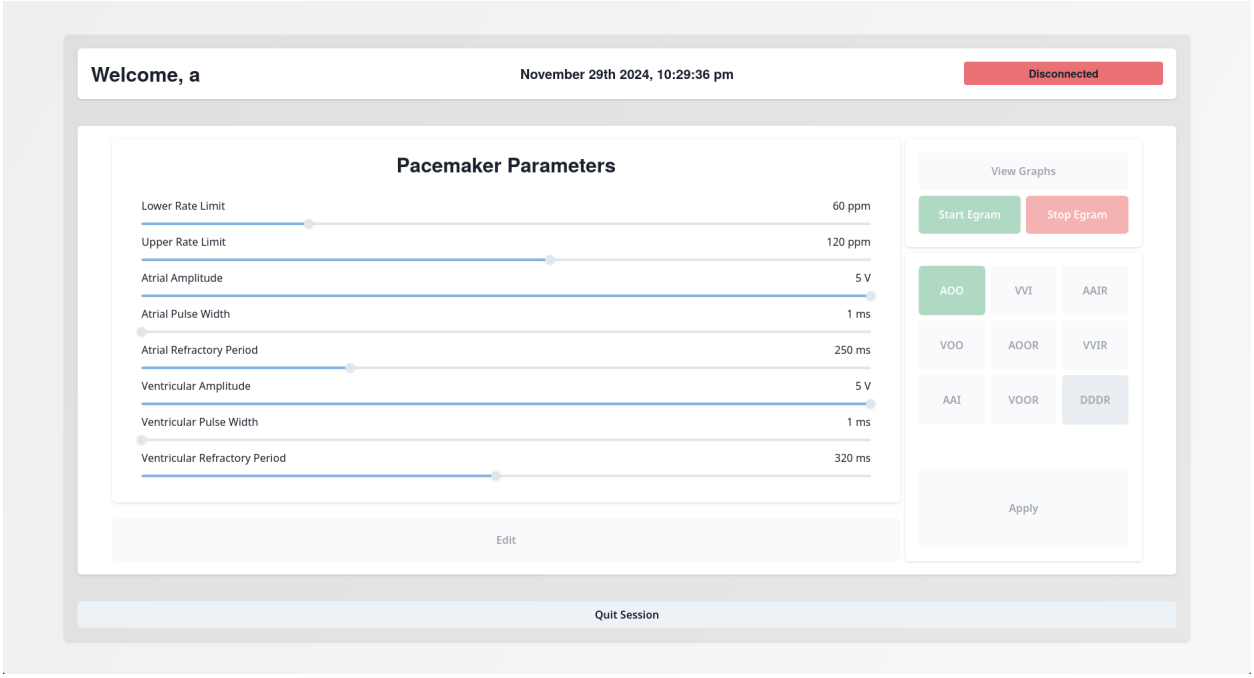


Figure 53: Buttons are grayed out when pacemaker disconnected

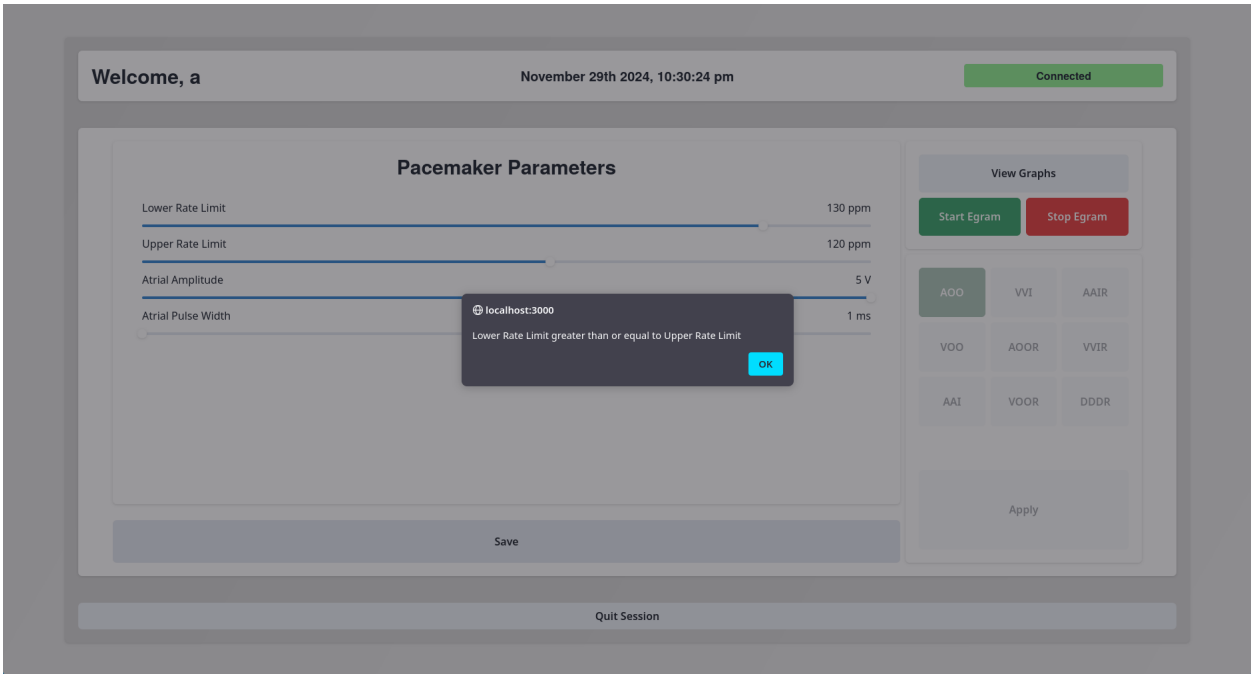


Figure 54: Result of setting LRL above URL

7 Serial Communications

UART (Universal Asynchronous Receiver/Transmitter), a serial communication protocol, enabled communication between the pacemaker and the DCM. It converts parallel data from a device into serial data for transmission and reconverts it to parallel data upon reception. Data is transmitted one bit at a time over a single data line, with timing governed by a specified baud rate. Each data packet is framed with start and stop bits, and the protocol operates asynchronously, meaning data is sent without requiring a shared clock signal.

A custom data packet format was defined to ensure reliable communication between the pacemaker and the DCM.

Table 59: Data Packet Format

Information	**Size (bytes)**	**Data Values/Range**	**Purpose**
SYNC	1	0x16	Used to signify the start of a payload (data).
Function Code	1	0x47, 0x49, 0x55, 0x88	Used to identify what action to take upon receiving a data packet. 0x47 – Send egram data to DCM 0x49 – Pacemakers sends parameters to DCM for verification

			0x55 – Update parameters on pacemaker with values sent from DCM 0x88 – Signifies that this data packet contains egram data
Mode	1	[0, 7]	To set the mode of the pacemaker. 1 = VOO, 2 = AOO, 3 = VVI, 4 = AAI, 5 = VOOR, 6 = AOOR, 7 = VVIR, 8 = AAIR, 9 = DDDR
Lower rate Limit	1	[30, 175]	To set corresponding pacemaker parameter.
Upper Rate Limit	1	[50, 175]	To set corresponding pacemaker parameter.
Atrial Amplitude	1	[0, 50]	To set corresponding pacemaker parameter. Value is atrial amplitude * 10
Atrial Pulse width	2	[1,30]	To set corresponding pacemaker parameter.
Atrial Refractory Period	2	[150, 500]	To set corresponding pacemaker parameter.
Ventricular Amplitude	1	[0, 50]	To set corresponding pacemaker parameter. Value is ventricular amplitude * 10.
Ventricular Pulse Width	2	[1, 30]	To set corresponding pacemaker parameter.
Ventricular Refractory Period	2	[150, 500]	To set corresponding pacemaker parameter.
Maximum Sensor Rate	1	[50, 175]	To set corresponding pacemaker parameter.
PVARP	2	[150, 500]	To set corresponding pacemaker parameter.
Hysteresis	1	[0, 175]	To set corresponding pacemaker parameter.
Rate smoothing	1	0, 3, 6, 9, 12, 15, 18, 21, or 25	To set corresponding pacemaker parameter.
Activity threshold	1	[1, 7]	To set corresponding pacemaker parameter. 1 = V-Low, 2 = Low, 3 = Med-Low, 4 = Med, 5 = Med-High, 6 = High, 7 = V-High
Reaction Time	1	[10, 50]	To set corresponding pacemaker parameter.
Response Factor	1	[1, 16]	To set corresponding pacemaker parameter.

Recovery Time	1	[2, 16]	To set corresponding pacemaker parameter.
Atrial Sensitivity	1	[0, 50]	To set corresponding pacemaker parameter. Value is atrial sensitivity * 10.
Ventricular Sensitivity	1	[0, 50]	To set corresponding pacemaker parameter. Value is ventricular sensitivity * 10.
Fixed AV delay	2	[70, 300]	To set corresponding pacemaker parameter.
Dynamic AV delay	1	[0, 1]	To set corresponding pacemaker parameter. 0 is off, 1 is on.
Minimum Dynamic AV Delay	1	[30, 100]	To set corresponding pacemaker parameter.
Sensed AV Delay Offset	1	[-100, 0]	To set corresponding pacemaker parameter.
PVARP Extension	2	[0, 400]	To set corresponding pacemaker parameter. 0 is off.
ATR Duration	2	[10, 2000]	To set corresponding pacemaker parameter.
ATR Fallback Mode	1	[0, 1]	To set corresponding pacemaker parameter. 0 is off, 1 is on.
ATR Fallback Time	1	[1, 5]	To set corresponding pacemaker parameter.
Checksum	1	N/A	Checksum is calculated by XORing every byte with each other. Used to enforce data integrity by detecting transmission errors due to interference.

A checksum bit is used to allow both the DCM and pacemaker to verify the integrity of the data packets.

8 Assurance Case

8.1 GSN

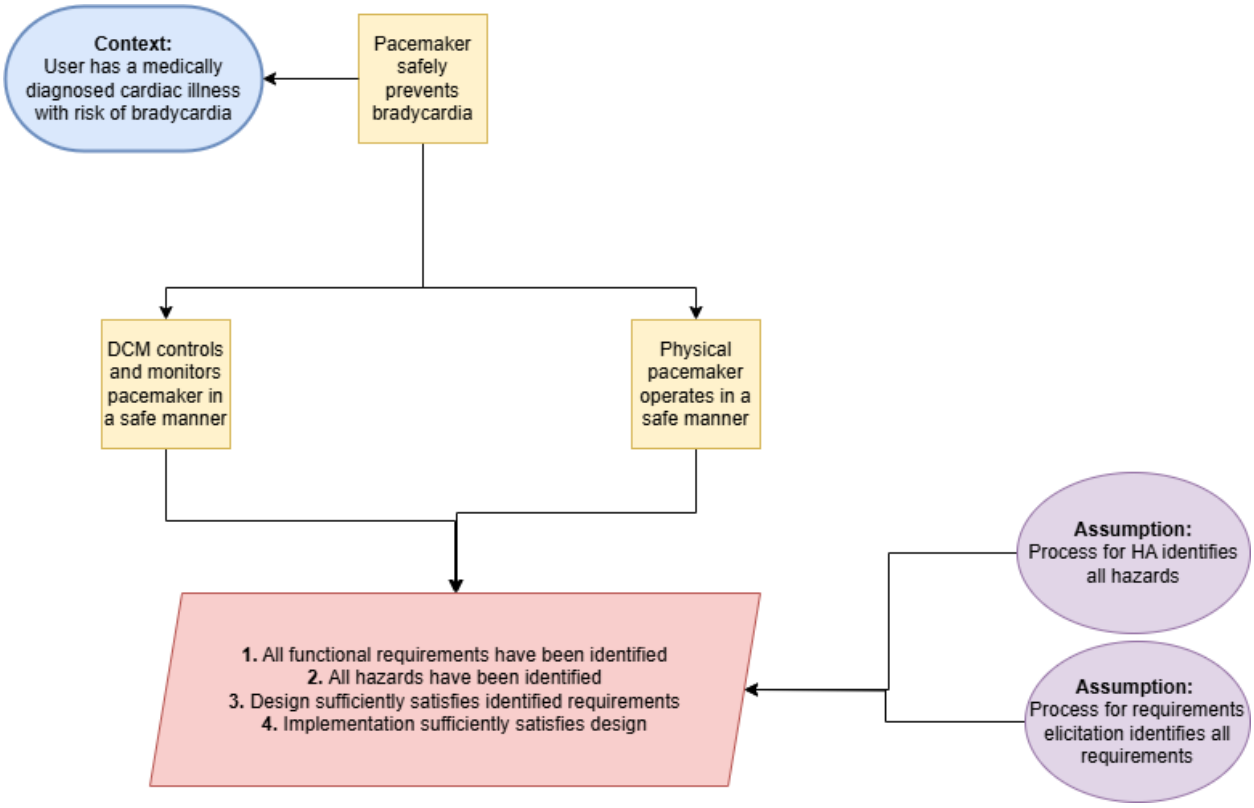


Figure 55: Top level GSN schematic

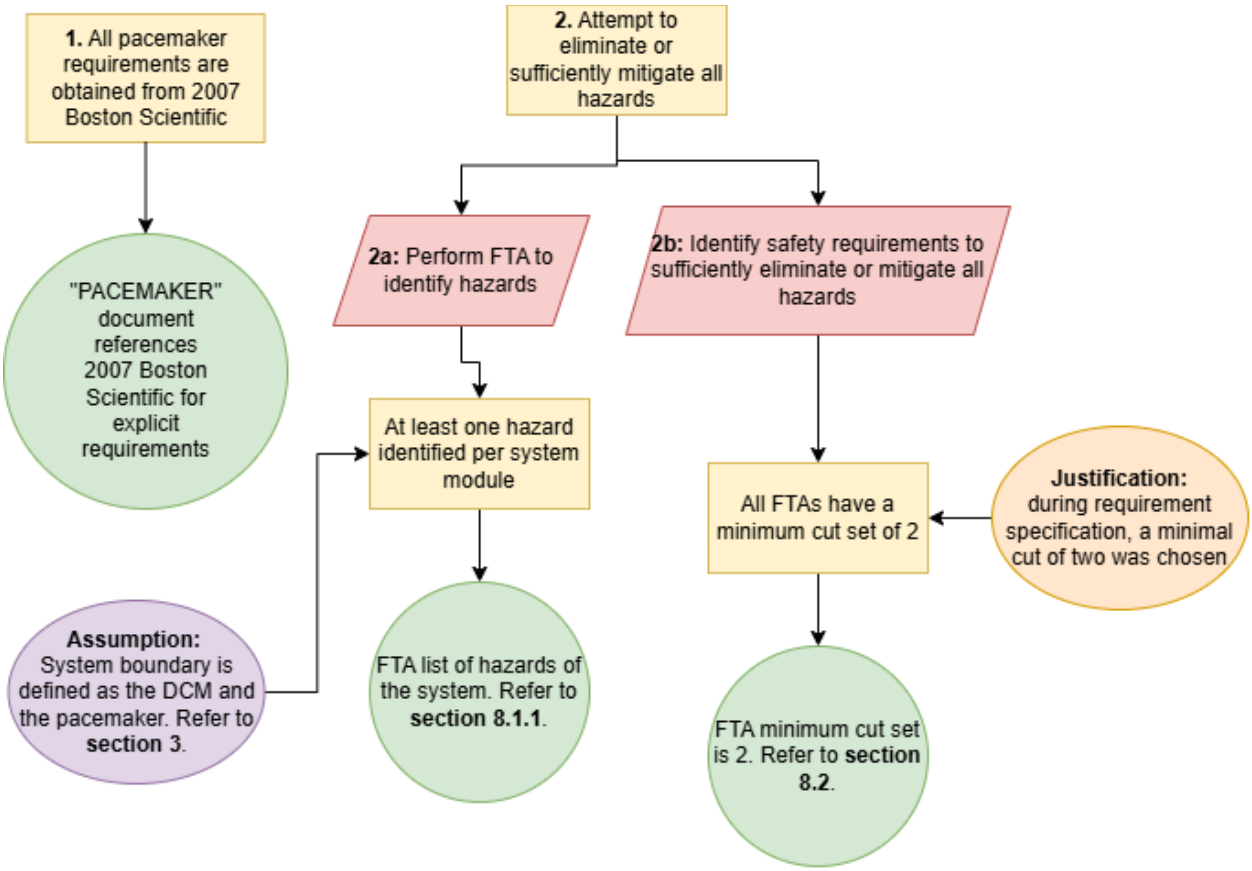


Figure 56: Bottom left level GSN schematic

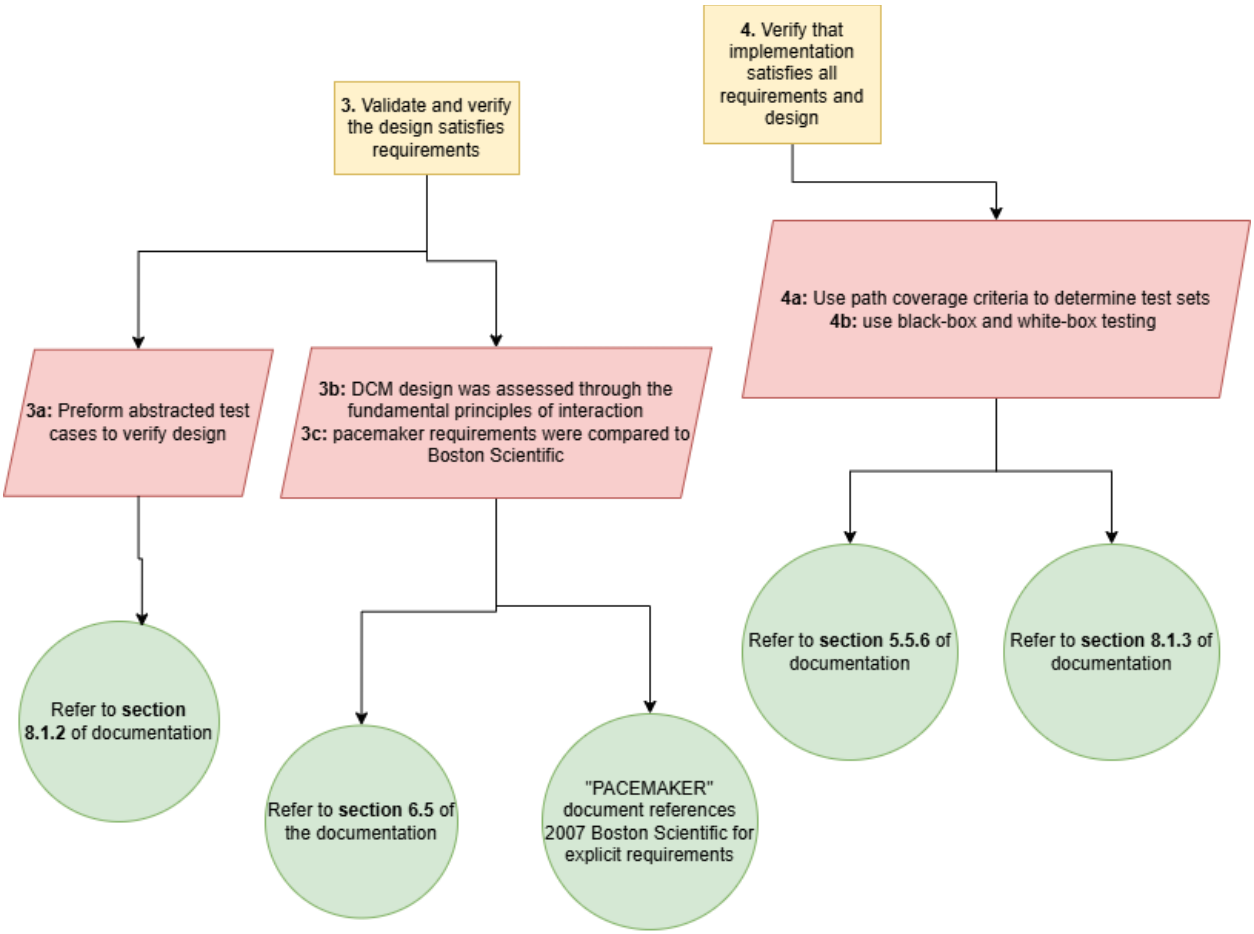


Figure 57: Bottom right level GSN schematic

8.1.1 2a Evidence

Major Hazards for the components of the system are identified:

- Pacemaker paces too slowly
- Pacemaker paces too quickly
- Pacemaker paces irregularly
- DCM parameter check fails

8.1.2 3a Evidence

For each mode, refer to the corresponding reference number.

Table 60: Reference table for 3a evidence.

Mode	Reference Number
Pacemaker System	**4**
VOO	**5.1.6**
AOO	**5.2.6**
VVI	**5.3.5**

AAI	**5.4.5**
Rate Adaptive Pacing	**5.5.6**
VOOR	**5.6.5**
AOOR	**5.7.5**
VVIR	**5.8.5**
AAIR	**5.9.5**

8.1.3 4b Evidence

For each mode, refer to the corresponding reference number.

Table 61: Reference table for 4b evidence.

Mode	Reference Number
Pacemaker System	**3**
VOO	**5.1.3, 5.1.4**
AOO	**5.2.3, 5.2.4**
VVI	**5.3.3, 5.3.4**
AAI	**5.4.3, 5.4.4**
Rate Adaptive Pacing	**5.5.3.2, 5.5.3.6, 5.5.4, 5.5.5**
VOOR	**5.6.3**
AOOR	**5.7.3**
VVIR	**5.8.3**
AAIR	**5.9.3**
DDDR	**5.10.3**

8.2 FTA

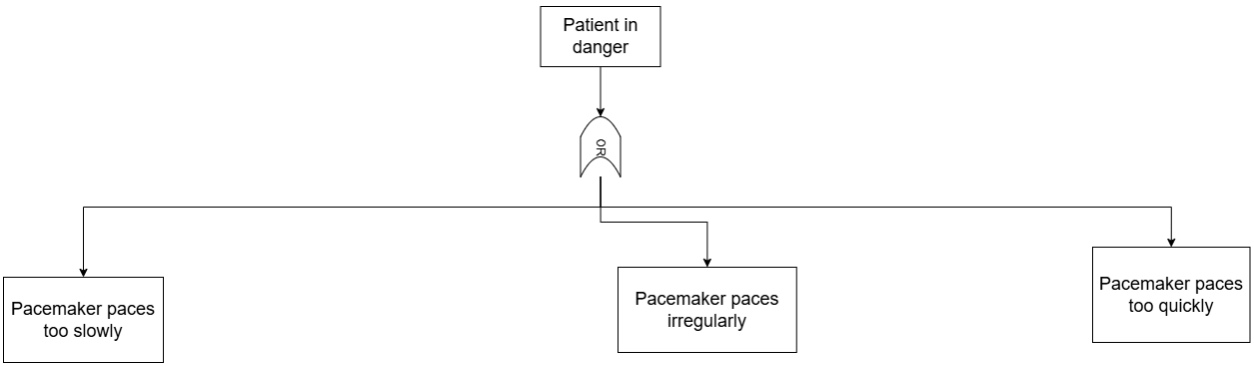


Figure 58: Top Level FTA

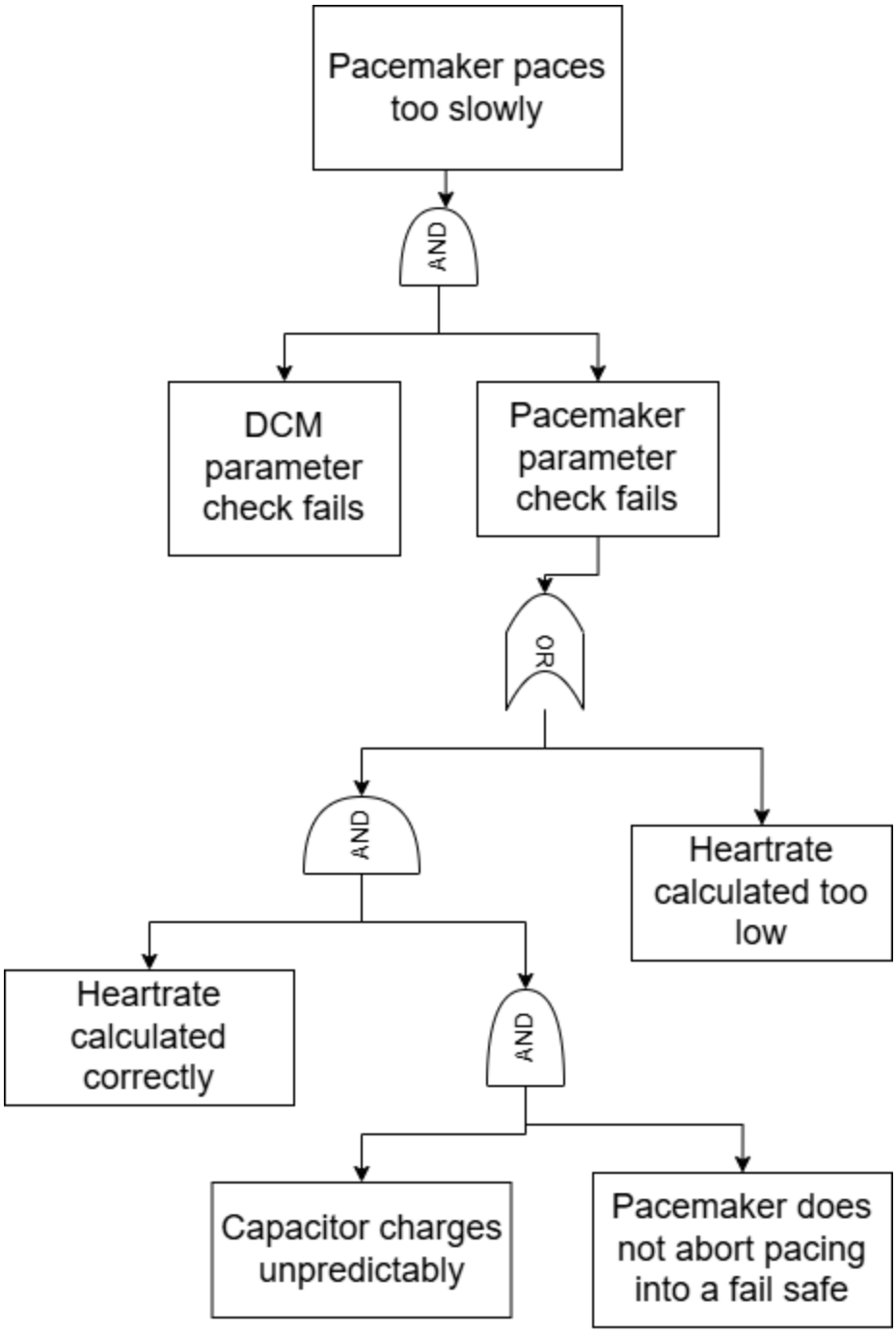


Figure 59: Bottom left level FTA

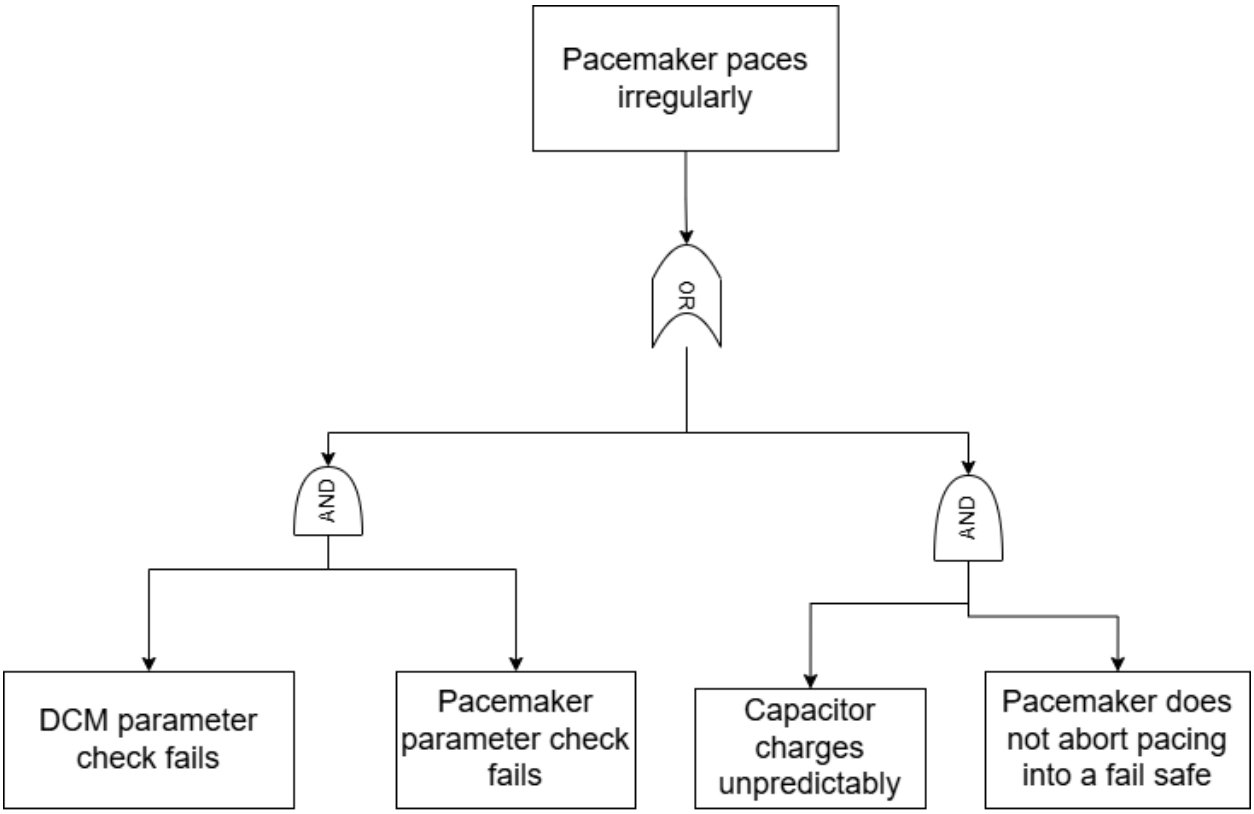


Figure 60: Bottom middle level FTA

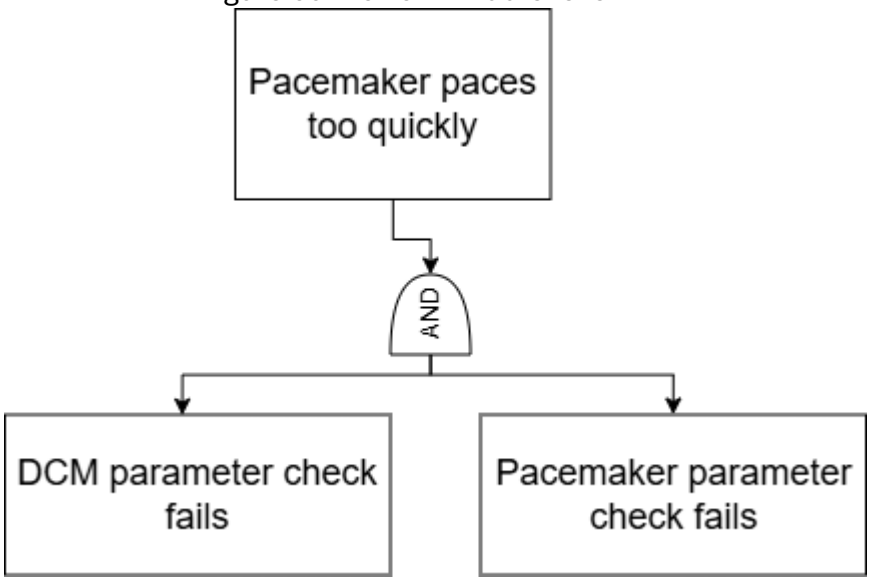


Figure 61: Bottom right level FTA

9 Journaling

The creation of this pacemaker began with the development of the VOO mode. The logic behind this mode was derived from initial pacemaker_shield_explained document. This laid the basis for the creation of the AOO mode. During the development of these modes, variances in the amplitude of the pacemaker's pulses was noticed. This was documented in an earlier prototyping version shown below.

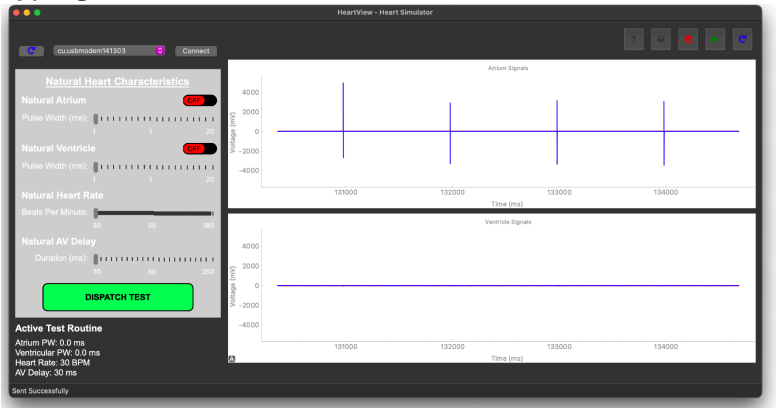


Figure 62: Screen capture of VOO mode.

Upon deliberation with the teaching assistants, it was evident that this issue was due to the unideal nature of the capacitors being charged and discharged.

Following the development of the VOO and AOO modes, the development of the VVI and AAI modes began. An earlier version of the VVI mode yielded the following results.

Figure 63: Screen capture of VVI mode attempt.

In the figure above, there seems to be a drop in voltage followed by a rapid increase in voltage amplitude. After some inspection of the Simulink states for this mode, it was determined that the pulsing states of this mode had the incorrect variables set to the incorrect values. Additionally, in this model, the after() function was used many times

within the transitions. However, these after() functions interfered with the VVI mode's ability to sense an incoming pulse. At the time, it was also unclear if the mode had actually waited the full refractory period before entering the sensing phase once again.

To create a more efficient VVI mode, the use of breakpoints and during action was implemented. The breakpoints helped debug the Stateflow diagrams and figure what the variables were being set to at each step of the way. Additionally, the after() functions were replaced with during actions that decremented a waiting time derived the pulse intervals and refractory periods. This method allowed the transitions to be solely used for conditional checks for if a natural heart pulse is sensed. At first this model did not work because the ventricular sensitivity was set to an extremely low amount that was not proportional to the desired duty cycle. Additionally, the reference pulse width modulation pin was set in a digital write block as opposed to an output pulse width modulation block. Essentially, the digitial write block was trying to write to a capacitor as opposed to charging it for comparison. Once this issue was resolved the VVI mode began pulsing and sensing effectively.

From there, the same logic was applied to develop the AAI mode. This mode did not differ much from the VVI mode. Following the development of these modes, hysteresis was explored. At first, a model of the VVI was created that included the hysteresis as a part of the countdown. However, this yielded results that we could not decipher. As a result, a different model was developed. This model included hysteresis as an independent state that was only entered following a sensed natural heart pulse that was followed by a timed out pulse interval. To ensure, this model worked, breakpoints were inserted within each state and transition. The values were closely montiored at each breakpoint. This helped in understanding exactly what is happening at each stage of the cycle.

After the four modes were finished, a combo box model was used to investigate how to implement mode switching. When choosing a mode, each mode had a number that, when selected, would handle the mode transitions between a central DEFAULT state. Once more, the use of breakpoints during testing and debugging ensured that the transitions worked with entering each of the different modes.

The use of information hiding was the last implementation. The input and output pins were gouped together and hidden, as were all the subsystems that were used to transform the constants into usable variables that we wanted to work with.

This development process lead to learning many lessons regarding troubleshooting. Especially the importance of laying out various breakpoints and debugging step by step.

9.1 Journaling November 29th, 2024

To create the adaptive rate pacing, an overview abstraction of the adaptive rate was first made which is pictured below.

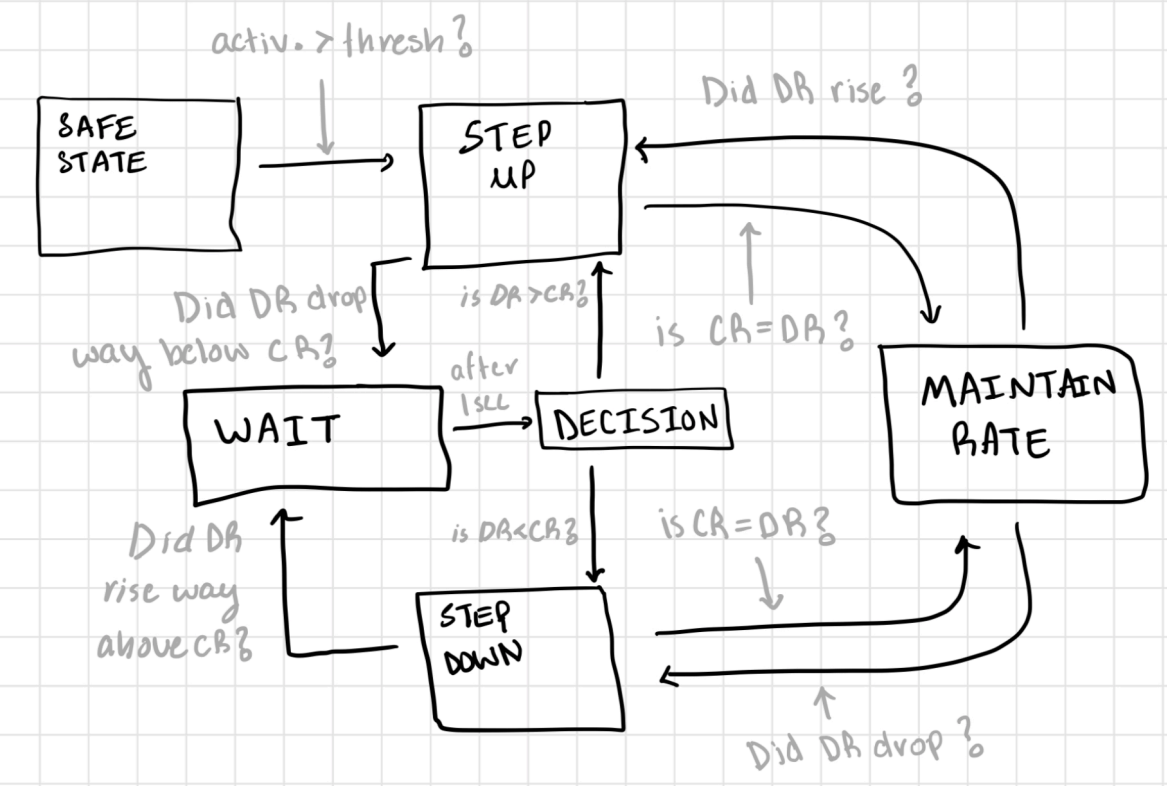


Figure 64: Overview abstraction of the adaptive rate pacing.

To create this abstraction, all the possible scenarios of rate adaptive pacing were first deduced. These states were the initial state, the stepping up of the rate, the maintaining of the pacing rate, and the stepping down of the rate. Moreover, the case of sudden increases or decreases were also analyzed. This abstraction helped in creating a basis for the Simulink model.

The adaptive pacing rate was taken from this model and used within the rate adaptive modes.

As for the serial communications, it was highly inspired by the tutorial four which showcased an example of serial communication between the pacemaker and the DCM.

To create the DDDR mode was also created using an overview abstraction that detailed the possible states of the dual chambers. This model takes into consideration the pacing of the two chambers, the sensing of both chambers, and the inhibition of either chamber. This is showcased in the figure below.

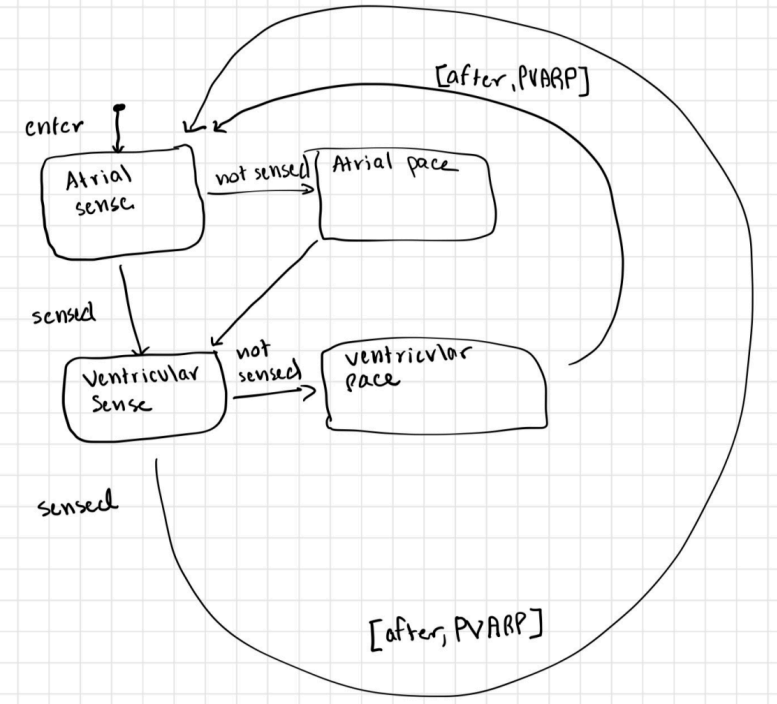


Figure 65: Overview abstraction of the DDDR.

The combination of these changes created the five additional modes as well as the serial communications between the pacemaker and the DCM.

9.2 Future Changes

9.2.1 Requirement Changes

October 25th, 2024
1. Must accommodate four more modes: VOOR, AOOR, VVIR and AAIR.
2. Must communicate to DCM to set parameters.
3. Must dynamically switch modes.
4. Must have precautionary set to bound parameters.
5. Must have checks to modulate time parameters if they overlap with other time parameters.

November 29th, 2024
1. Must incorporate magnet sensor for proximity.
2. Must fit into a smaller form factor.
3. Must be powered by a small battery.
4. Must implement security into DCM.

9.2.2 Design Decision Changes

October 25th, 2024
1. Hysteresis is no longer a programmable parameter and will correspond to the given lower rate limit.
2. The refractory period will become a dynamic parameter that responds to the natural heart rate.

November 29th, 2024
1. Pacemaker parameter safety checks are now global.