PACEMAKER DOCUMENTATION

3K04 Assignment 2

Asheel Dowd - 400470770
Anna Naumova - 400458754
Daniel Chirackal - 400452897
Jaavin Mohanakumar - 400465588
Jacob Foster - 400474753
Tyler Smith - 400435424

Table of Contents

System Overview

Commented [TS1]: Needs to be updated for A2 (can talk more about serial, integration of the two, etc)

Pacemakers are safety critical systems that provide electrical stimulation to the chambers of the heart to treat arrhythmias or irregular heartbeats. Our system consists of the pacemaker hardware kit and the Device Controller-Monitor (DCM). The DCM acts as the human interface for the physical pacemaker hardware. This allows for care to be prescribed to the patient by designated healthcare professionals, such as selecting different pacing rates and modes. The pacemaker implements the care prescribed through the DCM by performing functions such as pacing the atrium and ventricle of the heart and sensing natural pulses.

The Pacemaker features adaptive rates that are enforced by the users' physical movements allowing for dynamic changes in pulsing, in other words the device accounts for the day-to-day tasks of the patient. The DCM interfaces with the pacemaker's hardware allowing the showcase of critical data like the patient's real time heart activity via an ECG (Electrocardiogram) graph. The DCM allows for the viewing of real time pacemaker activity, showing inhibited, and regular pluses. All of this is done via seamless UART serial commutation, allowing the sending and receiving of Pacemaker parameters from either side.

The DCM serves as a critical interface between healthcare providers and the health of patients utilizing the pacemaker as it allows for the safe and secure storage and verification of pacemaker operational parameters. This allows doctors and other healthcare professionals to easily and securely provide aid to patients using the pacemaker system.

Design

Requirements

DCM

The DCM must handle user sign-ins and registrations but not allow more than ten users and their data to be stored locally. The system must protect (ie. hashing) user passwords in case of a compromised storage device. The DCM must hold the data entered by the user for the four pacing modes. The interface must display the programmable data inputted by the user for each pacing mode and project it on the corresponding electrogram. The front end must also inform the user when the DCM and Pacemaker communicate and their status. The DCM must convey if a new Pacemaker (different than the one previously used) approaches the device. The UI must be operable by any healthcare professional, regardless of tech literacy.

Pacemaker

Commented [TS2]: Additional details on serial and rate adaptive additions for assignment 2

The pacemaker is to provide pacing to the heart in eight modes as specified throughout. These are AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR and VVIR. The pacemaker must have the ability to sense pulses on both the atrium and ventricle. The pacemaker must be able to provide pacing to both the atrium and ventricle of varying amplitudes, pulse widths, and rates. The pacemaker must be able to utilize a variable refractory period to disregard subsequent pulses. The pacemaker must be able to utilize the internal accelerometer to adapt the rate of pacing based on a set response time, recovery time, activity threshold, response factor and maximum sensor rate. The pacemaker must be able to receive parameters over UART and set them for use in operation. The pacemaker must be able to send the currently active parameters over UART. The pacemaker must be able to sample atrium and ventricle activity and send it over UART for use in graphing. The pacemaker must ensure all parameters are within acceptable ranges at all times.

AOO

In AOO mode, the atrium is paced at a constant rate that is externally determined, regardless of heart activity. No sensing, ventricular pacing, or rate adaptation is performed.

VOO

In VOO mode, the ventricle is paced at a constant rate that is externally determined, regardless of heart activity. No sensing, atrium pacing, or rate adaptation is performed.

AAI

In AAI mode, the atrium is paced at a rate that is externally determined. While a constant rate is set, generated pulses are inhibited if a (presumably naturally generated) pulse is detected in the atrium. No ventricle pacing or rate adaptation is performed. An atrial refractory period allows subsequently detected pulses to be ignored in the case that naturally generated pulses are detected.

VVI

In VVI mode, the ventricle is paced at a rate that is externally determined. While a constant rate is set, generated pulses are inhibited if a (presumably naturally generated) pulse is detected in the ventricle. No atrium pacing or rate adaptation is performed. A ventricular refractory period allows subsequently detected pulses to be ignored in the case that naturally generated pulses are detected.

AOOR

In AOOR mode, the atrium is paced at a rate that is determined by the accelerometer data from the pacemaker, regardless of any natural atrium paces. No sense inhibiting or ventricular pacing is performed.

VOOR

In VOOR mode, the ventricle is paced at a rate that is determined by the accelerometer data from the pacemaker, regardless of any natural ventricle paces. No sense inhibiting or atrium pacing is performed.

AAIR

In AAIR mode, the atrium is paced at a rate that is determined by accelerometer data from the pacemaker. Generated pulses are inhibited if a (presumably naturally generated) pulse is detected in the atrium. No ventricle pacing is performed. An atrial refractory period allows subsequently detected pulses to be ignored in the case that naturally generated pulses are detected.

VVIR

In VVIR mode, the ventricle is paced at a rate that is determined by accelerometer data from the pacemaker. Generated pulses are inhibited if a (presumably naturally generated) pulse is detected in the ventricle. No atrium pacing is performed. An ventricle refractory period allows subsequently detected pulses to be ignored in the case that naturally generated pulses are detected.

Implementation – Pacemaker

Hardware + Variables

Our input variables for the main pacing stateflow are listed below.

Name	Source	Functional Description	Value Range	Units
Mode	User specified (DCM)	The mode is an enumeration used to specify which operating mode the pacemaker should be operating in. 1 = AOO, 2 = VOO, 3 = AAI, 4 = VVI, 5 = AOOR,	1-8	Enumeration

Commented [TS3]: Needs to be completed for these 4 modes.

Commented [TS4]: Add new variables for assignment 2, remove asterisks and specify from DCM.

Commented [AN5R4]: Would be nice to put upper and lower bounds of each as well, or like values (ex. 1-16 for response factor)

		6 = VOOR, 7 = AAIR, 8 = VVIR.		
LRL	User specified (DCM)	The lower rate limit doubles as the specified BPM for pacing for non rate adaptive modes. Used in all modes.	30-175	BPM
URL	User specified (DCM)	Sets the maximum allowed pacing rate for the pacemaker.	50- 175	BPM
ATR_pulseAmplitude	User specified (DCM)	The atrial pulse amplitude specifies the amplitude of atrial pulses in volts. Used in AOO, AOOR, AAI and AAIR modes.	0.5-5	Volts
ATR_pulseWidth	User specified (DCM)	This specifies the pulse width in milliseconds for pulses of the atrium. Utilized in AOO, AOOR, AAI and AAIR modes.	0.05- 1.9	ms
VENT_pulseAmplitude	User specified (DCM)	The ventricle pulse amplitude specifies the amplitude of ventricle pulses in volts. Used in VOO, VVI, VOOR and VVIR modes.	0.5-5	Volts
VENT_pulseWidth	User specified (DCM)	This specified the pulse width in milliseconds for pulses of the ventricle. Used in VOO, VVI, VOOR and VVIR modes.	0.05- 1.9	ms
VRP	User specified (DCM)	The ventricular refractory period specifies the period of time in milliseconds that follows a sensed pulse on the ventricle, in which a detection will not result in an inhibition of the next pulse. Used in the VVI mode.	150- 500	ms

ARP	User specified (DCM)	The atrial refractory period specifies the period of time in milliseconds that follows a sensed pulse on the atrium, in which a detection will not result in an inhibition of the next pulse. Used in the AAI mode.	150- 500	ms
MaxSensorRate	User specified (DCM)	Used to set the maximum pacing rate output by the rate adaptive/accelerometer block.	50- 175	BPM
ReactionTime	User specified (DCM)	The reaction time is the time it takes for the rate to change from the LRL to the MSR in rate adaptive pacing. Used to set the rate of BPM increase in rate adaptive pacing.	10-50	seconds
ResponseFactor	User specified (DCM)	The response factor determines the “responsiveness” of the activity level based on the accelerometer data.	1-16	Unitless
RecoveryTime	User specified (DCM)	The recovery time is the time it takes for the rate to change from the MSR to the LRL in rate adaptive pacing. Used to set the rate of BPM decrease in rate adaptive pacing.	2-16	minutes
ActivityThreshold	User specified (DCM)	The activity threshold it the minimum activity level that triggers a response in the target rate.	Very Low, Low, Med-Low, Med, Med-High, High, V-High	Enumeration
ATR_CMP_DETECT	Hardware (Pin D0)	The atrial comparator detect signal is a rising edge on detection of an	0, 1	Boolean

		atrial pulse. Used for sensing in AAI mode.		
VENT_CMP_DETECT	Hardware (Pin D1)	The ventricle comparator detect signal is a rising edge on detection of an ventricular pulse. Used for sensing in VVI mode.	0, 1	Boolean

Table 1: Input variables and their uses.

Figure 1: Simulink Input Structure

What is shown above is the revised Simulink input structure. Once the parameters are sent to the pacemaker (see the serial communication section), and typecasted according to the appropriate data types, via the input block, they are passed through a saturation block. The saturation block caps values for certain parameters that are above the upper limit at the upper limit, in similar fashion the same process happens for values lower than the lower limit. These values are connected to the main state flow via tags that seamlessly transfer the values.

Our output variables are listed below.

Commented [TS6]: Don't think we have any new output variables to the heart, if so add here

Commented [AN7R6]: No new ones

Name	Destination	Functional Description
Z_VENT_CTRL	Hardware (Pin D7)	Controls the connection of the ventricle to the impedance circuit.
ATR_PACE_CTRL	Hardware (Pin D8)	This signal is used to discharge the charging capacitor into the atrium, used in pacing the atrium (modes AOO and AAI).

PACE_CHARGE_CTRL	Hardware (Pin D2)	This signal is used to charge the charging capacitor. The charging capacitor is charged in between paces for all states.
PACE_GND_CTRL	Hardware (Pin D10)	This controls the connection from the heart ground rind to ground. Used to pace either chamber of the heart. Used in all modes.
VENT_PACE_CTRL	Hardware (Pin D9)	This signal is used to discharge the charging capacitor into the ventricle, used in pacing the ventricle (modes VOO and VVI).
VENT_GND_CTRL	Hardware (Pin D12)	Used to discharge the blocking capacitor back into the ventricle in conjunction with charging, used in VOO and VVI.
ATR_GND_CTRL	Hardware (Pin D11)	Used to discharge the blocking capacitor back into the atrium in conjunction with charging, used in AOO and AAI.
Z_ATR_CTRL	Hardware (Pin D4)	Controls the connection of the atrium to the impedance circuit.
FRONTEND_CTRL	Hardware (Pin D13)	This signal enables sensing of pulses on both the atrium and ventricle. Used in all modes.
PACING_REF_PWM	Hardware (Pin D5)	This is an output PWM signal that is used to charge the charging capacitor to the proper voltage level before discharging into the heart.
VENT_CMP_REF_PWM	Hardware (Pin D3)	This is an output PWM signal that sets the comparator voltage that is used to sense whether a pulse has been detected on the ventricle.
ATR_CMP_REF_PWM	Hardware (Pin D6)	This is an output PWM signal that sets the comparator voltage that is used to sense whether a pulse has been detected on the atrium.

Table 2: Output variables and their uses.

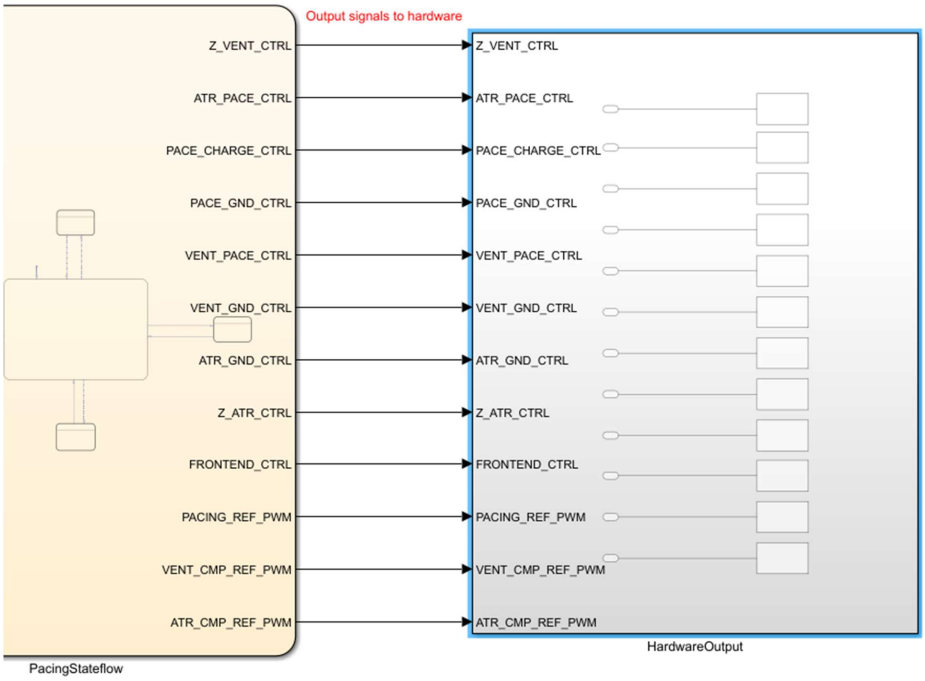


Figure 2: Output variables and hardware subsystem.

The output variables from the stateflow chart are fed into a hardware output subsystem. We implement hardware hiding using this subsystem, allowing the hardware complexities to be abstracted away so we can focus on the signals and their meaning to the Simulink code. This also allows for a modular design. The logic of the pacemaker could be changed completely and written in a new stateflow chart. However, that new logic could simply be connected to this same subsystem without issue. Because the subsystem has one specific function, this allows for it to be decoupled from the rest of the logic, while displaying cohesion by including all of the hardware outputs.

Sensing and Pacing Implementation

The core logic of the pacemaker is broken up into eight blocks corresponding to each of the eight states.

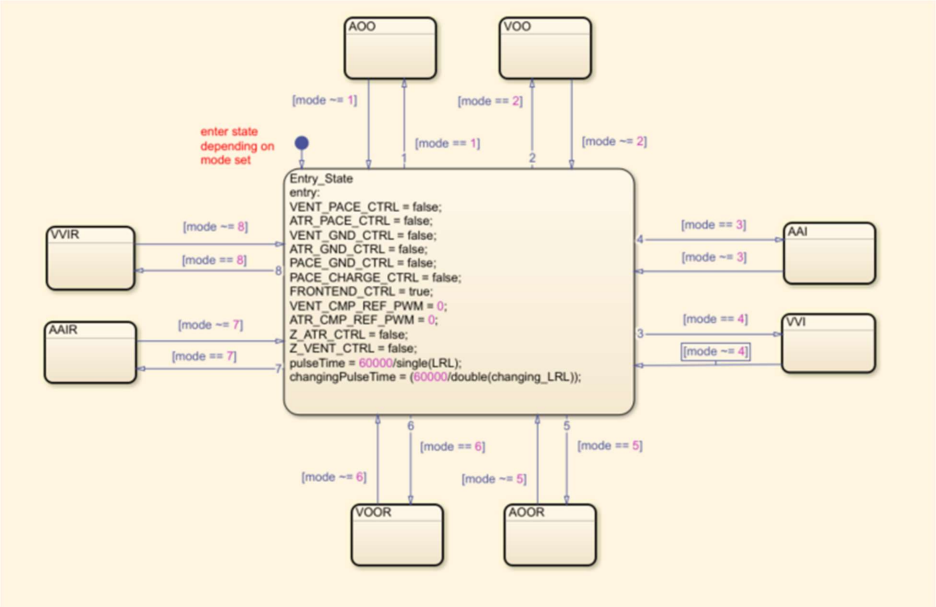


Figure 3: Top-level Siumlink stateflow

Commented [TS8]: Needs to reflect updated top level stateflow

You can see the behaviour of the mode input variable as an enumeration that selects which mode will be active based on the parameter set from the DCM. All local pacing variables are reset in the entry state to ensure that all states are starting from a clean slate and changes to outputs in one mode will not affect the operation of subsequent modes. Additionally, the *pulseTime* variable is calculated. This is 60 000 divided by the LRL (specified BPM). This division converts a BPM to the period in milliseconds for a full charge and pace cycle. The variable *changingPulseTime* is also calculated based on the same logic as above. However, instead of the lower rate limit being used as the set BPM the input variable *changing_LRL* is used instead. This input is the output from the rate adaptive pacing block as described in *Rate Adaptive Pacing Implementation*. The logic for each of the modes is described here.

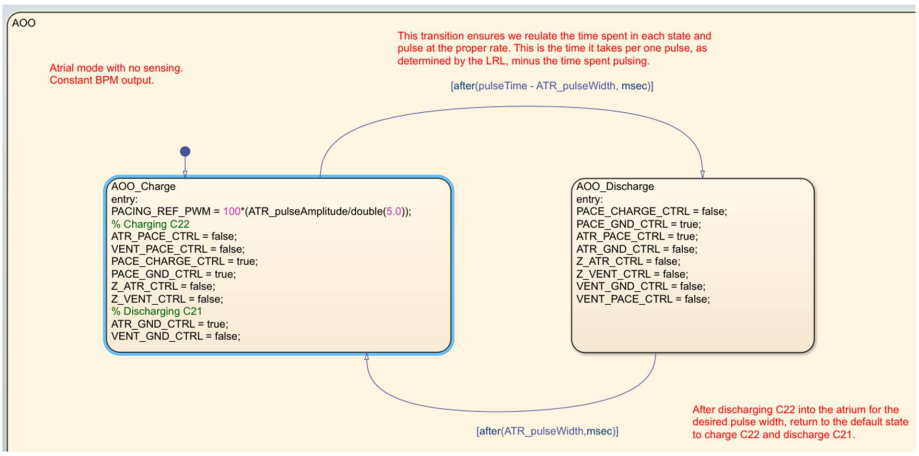


Figure 4: Mode 1 – AOO.

The AOO code is a good place to highlight the charging and discharging processes. The AOO_Charge state is responsible for applying voltage across the charging capacitor while also discharging the blocking capacitor back into the atrium to prevent charge buildup. You can see the code for these two steps split by comments within the state. The function for each of these output variables that are being set can be found in the *Hardware + Variables* section.

PACING_REF_PWM is responsible for setting the proper voltage on the charging capacitor and therefore impacts the amplitude of the output pulses. Because the output is scaled between 0V and 5V, we set a duty cycle percentage to scale that voltage (where 100% = 5V, 50% = 2.5V, 0% = 0V, etc.).

The AOO_Discharge state discharges the charging capacitor into the atrium. The signals being set and their functions can be found in the *Hardware + Variables* section.

The state transitions are responsible for AOO delivering a constant steady pacing to the atrium. A transition from the discharge to charge state occurs after the pulse width (specified in milliseconds) has elapsed. This results in the charging capacitor discharging for the duration specified as the pulsewidth. The transition from charge to discharge occurs after a time specified as the full pacing period minus the pulsewidth has elapsed. This ensures pacing continues occurring at a constant rate specified by the pull pacing period (and indirectly the specified BPM).

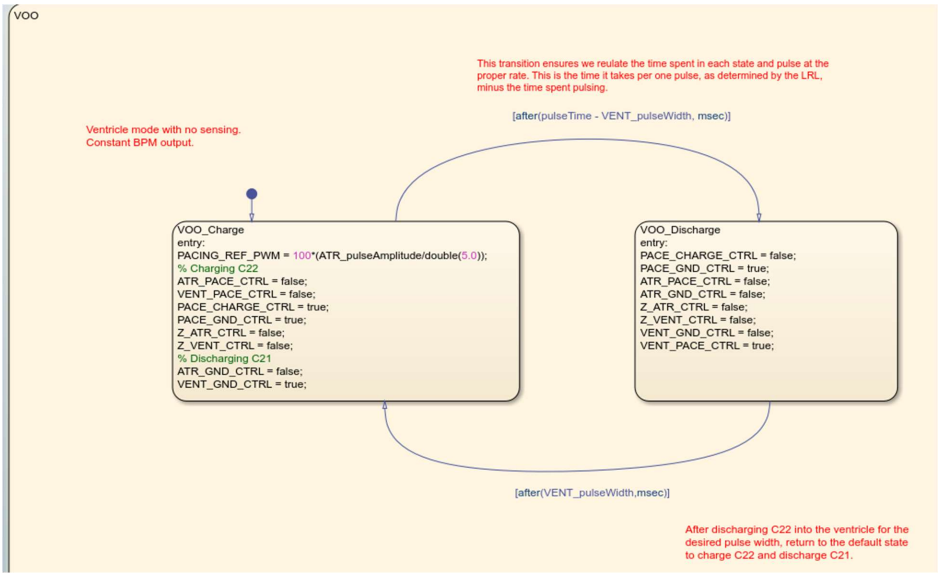


Figure 5: Mode 2 – VOO.

The VOO mode is identical in terms of logic to the AOO mode. The only difference is that discharging the charging capacitor and blocking capacitor both occur into the ventricle rather than the atrium.

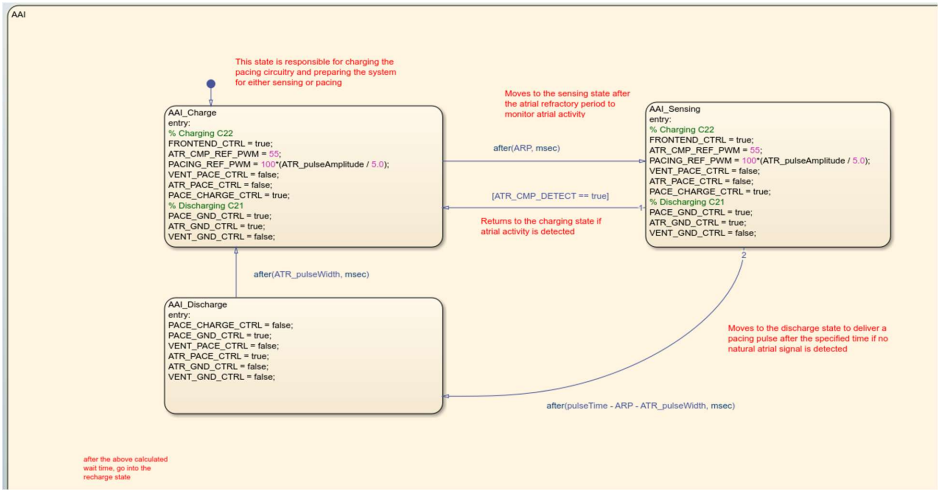


Figure 6: Mode 3 - AAI.

In AAI, the process of discharging the charging capacitor into the atrium is identical to that seen in AOO. The charging process is similarly identical, with the addition of setting FRONTEND_CTRL to true, effectively enabling sensing. The atrial comparison voltage was empirically set to 2.75V to ensure that false reads are mitigated (as would occur with too low of a comparison voltage) and all pulses are detected (which would not occur if with too high of a comparison voltage).

The transitions are altered from the AAI in order to account for pulse inhibition in the event of a pulse detection in the atrium. In the AAI_sensing state, the charging actions from AOO are performed (charging the charging capacitor and discharging the blocking capacitor). However, in the event of a pulse being detected in the atrium, the program transitions into the AAI_charging state. The same actions are performed in this state are the same as the AAI_sensing state, and it remains in this state for a time specified as the atrial refractory period. This prevents subsequent pulse detections from continuously inhibiting the pacing.

After the refractory period has elapsed, the AAI_sensing state is re-entered. If, instead of a pulse being detected, a time equal to the full pacing period minus the refractory period and pulse width elapses, the atrium is paced. This is done using the same logic as AOO.

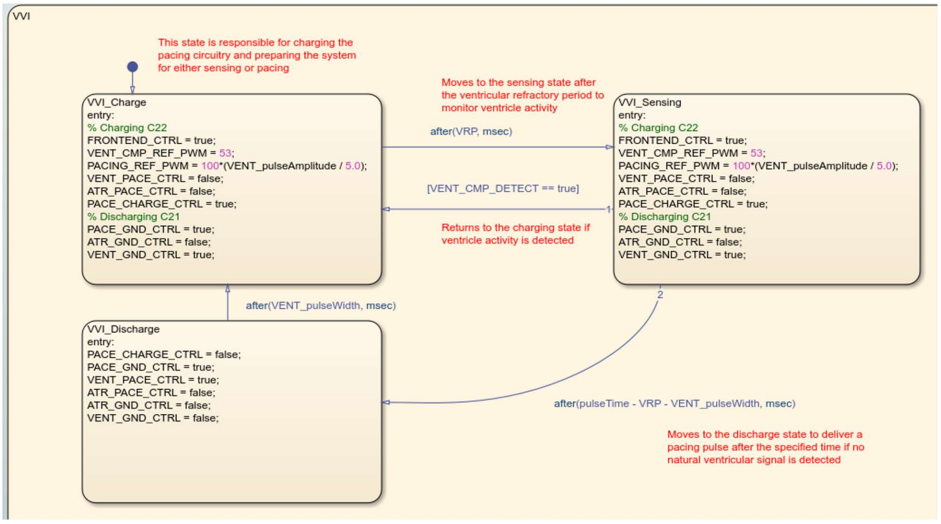


Figure 7: Mode 4 – VVI.

The VVI mode is identical in terms of logic to the AAI mode. The only difference is that discharging the charging capacitor and blocking capacitor both occur into the ventricle rather than the atrium. Additionally, the ventricular refractory period and pulse widths are used, while sensing occurs on the ventricle rather than atrium.

The rate adaptive modes take a nearly identical form to their non-adaptive counterparts. They are described below.

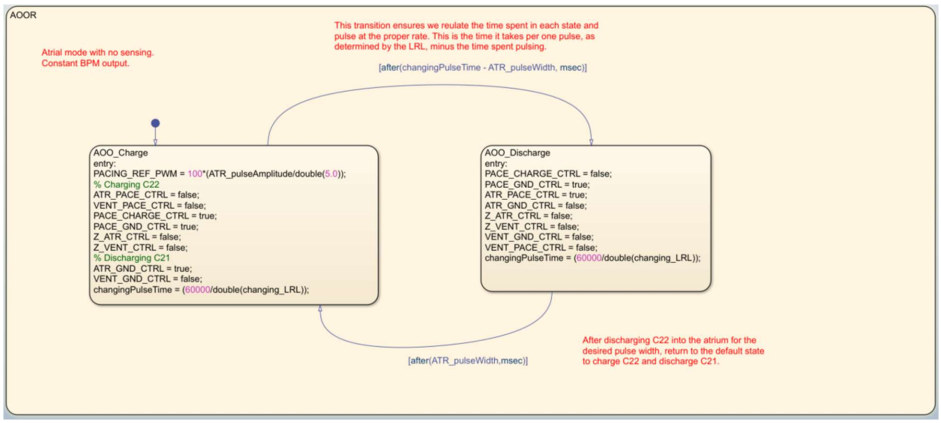


Figure 8: AOOR.

The AOOR features the same charging and discharging logic seen in the AOO mode, with the noticeable difference being the use of *changingPulseTime* as opposed to *pulseTime* along with the calculation of that same variable. Calculation of *changingPulseTime* is required continuously as the variable that drives it, *changing_LRL* is the output from the rate adaptive block which changes in real time.

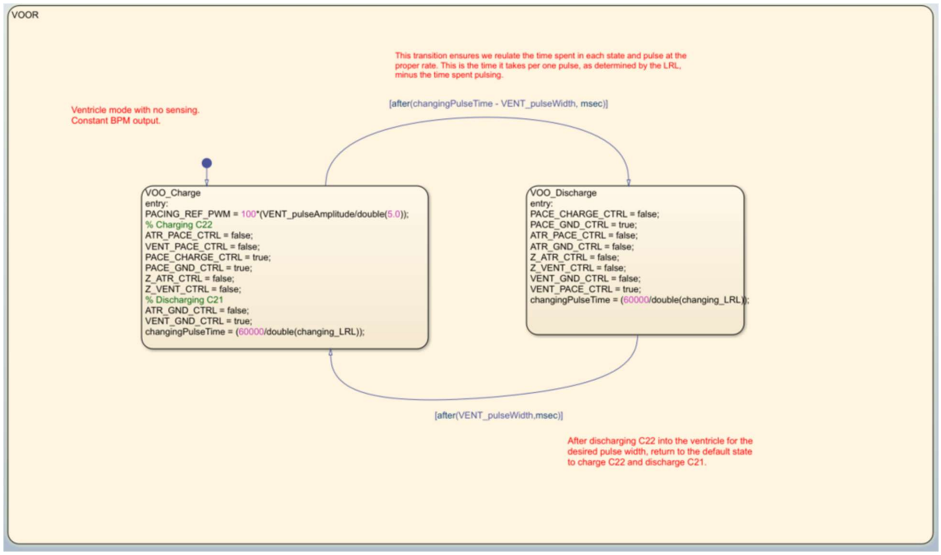


Figure 9: VOOR.

Like AOOR, VOOR is identical to its non-adaptive counterpart. The same arguments regarding *changingPulseTime* utilized for AOOR hold for this mode.

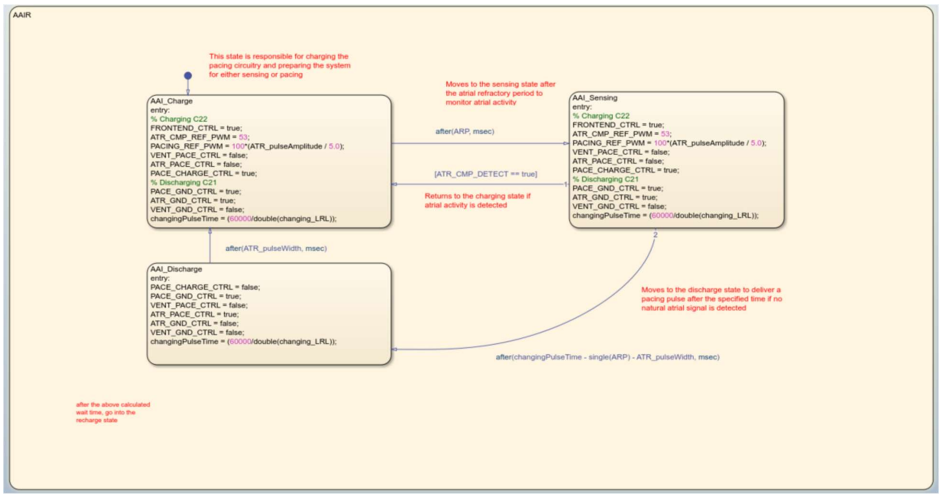


Figure 10: AAIR.

AAIR has the same logic as AAI with the exception of the *changingPulseTime* variable as described above.

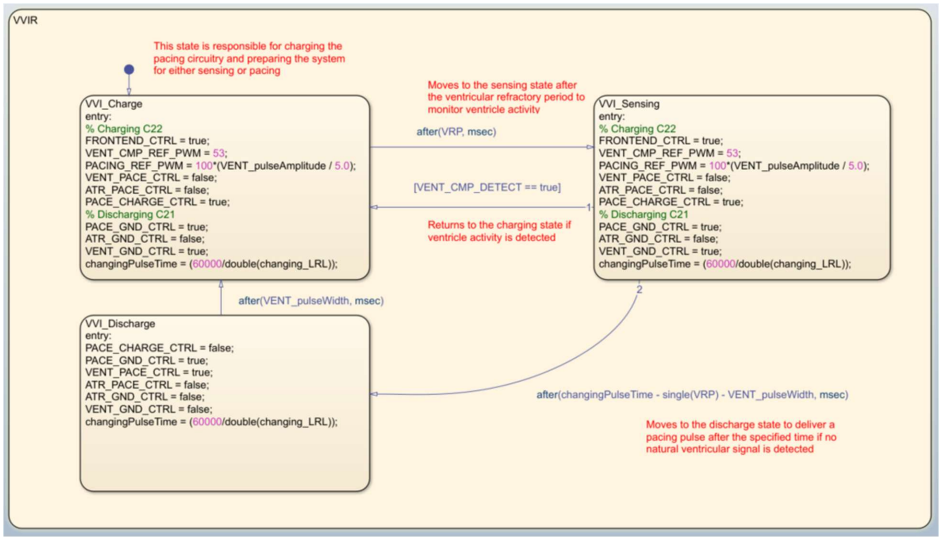


Figure 11: VVIR.

VVIR has the same logic as VVI with the exception of the *changingPulseTime* variable as described above.

Rate Adaptive Pacing Implementation

Rate adaptive pacing is a feature of pacemakers designed to adjust the pacing rate of the heart based on the body's activity level. It ensures that the heart rate matches the demands of the body during activities like exercise or emotional distress. Due the inclusion of rate adaptive pacing modes AOOR, VOOR, AAIR, and VVIR, this Simulink implementation included a new module to handle rate adaptive pacing. The main output is an adapted rate used in the main pacing state flow. The rate adaptation module is composed of three parts: an accelerometer subsystem, a chart used to convert activity level to a threshold, and the rate adaptation calculation subsystem.

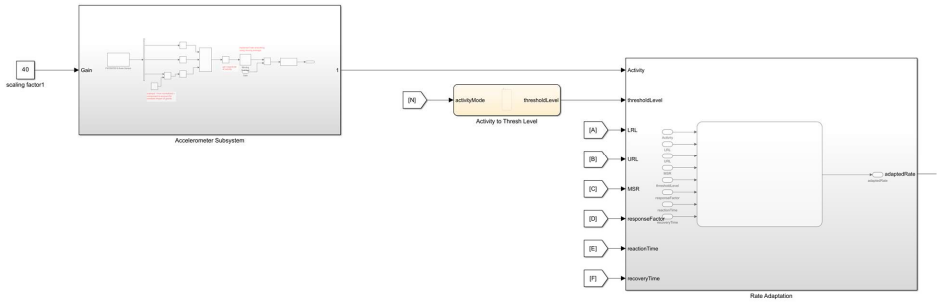


Figure 12: Rate Adaptive Module

Accelerometer Subsystem

The accelerometer subsystem, shown in Figure 13, is responsible for translating raw data into a processed activity signal that captures the overall motion. The inputs into this subsystem include a scaling factor and the signal from the FRDM accelerometer located on the pacemaker, and the output is the activity. It operates in three main stages with the first being magnitude calculation. Once the 3-axis accelerometer input vector is obtained from the accelerometer, the vector is split up into three components, x, y, z, which are all normalized. One is subtracted from the z-component to account for the constant impact of gravity. The magnitude of the activity is then calculated using the vector magnitude formula. His ensures that the activity signal reflects motion in any orientation.

The next step is rate smoothing. The single scalar vector is passed into a moving average block which takes the average of the signal with a sliding window. This helps avoid the impact of short, impulse-like movements effectively smoothing the signal. The moving average method was chosen for its simplicity and ability to provide continuous updates.

In the last stage, scaling, the smooth activity signal is multiplied by a scaling factor to adjust its range, making it suitable for subsequent threshold and rate adaptive calculations. The scaling factor was determined through iterative testing to achieve optimal range with expected activity

thresholds. Finally the signal is converted to an unsigned 8 bit integer format ensuring compatibility with downstream systems.

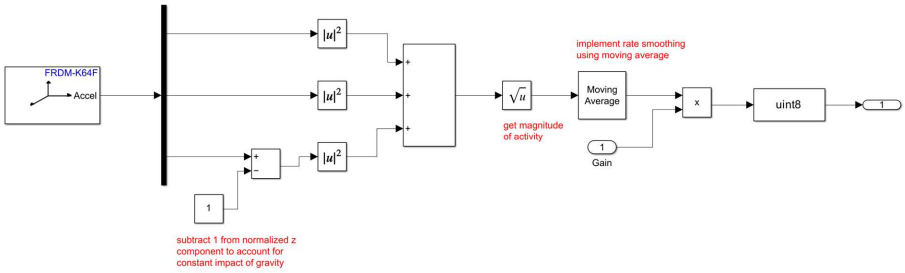


Figure 13: Accelerometer Subsystem

Activity to Threshold Level Converter

The activity to threshold level converter can be seen in Figure 14 and the corresponding state flow can be seen in Figure 15. The converter takes an input from the DCM called *activityMode*. It is an enumerated constant representing different operational modes for activity. This mode transmitted over serial or converted into their corresponding threshold values using multiple if statements. These output threshold levels were determined empirical testing and can be seen in Table 3.

Table 3: Summary of Threshold Level Values for Each Mode

Activity Threshold Mode	Enumeration Value	Activity Threshold Level
V-Low	0	5
Low	1	13
Med-Low	2	21
Med	3	29
Med-High	4	37
High	5	45
V-High	6	53

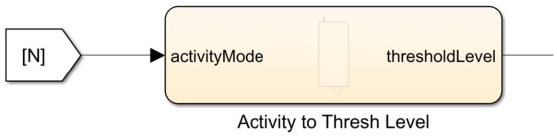


Figure 14: Activity to Threshold Level Subsystem

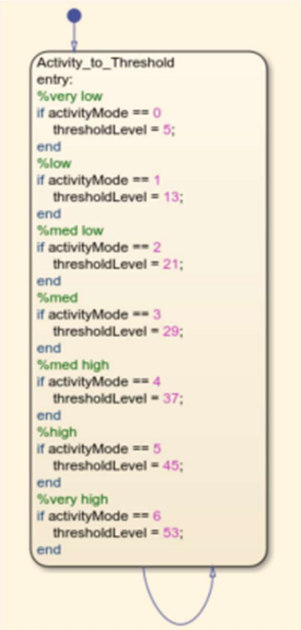


Figure 15: Stateflow Used to Obtain Threshold Level

Rate Adaptive Calculation Subsystem

The adaptation subsystem, Figure 16, calculates the pacing rate based on activity levels and user defined parameters on the DCM. As can be seen in Figure 17, the stateflow begins by calculating the slope which determines the target rate increase per unit increase in activity. This slope is proportional to the response factor allowing higher RF values to induce steeper responses to changes in activity. This slope is calculated using the equation,

$$slope = 0.75 * RF$$

where the constant 0.75 was empirically determined to help scale the rate factor. Lower rate limit and maximum sensor rate define the rate boundaries, and the threshold is used as the activity baseline. Using this slope the target rate is derived from

$$target = LRL + (slope * (Activity - Threshold))$$

The target rate is constrained within the lower rate limit, upper rate limit, and maximum sensor rate to ensure safety. If the calculated rate exceeds the MSR or URL, or falls below the LRL, it is bounded accordingly. Next, the subsystem calculates the steps required to increment, *stepReaction*, or decrement, *stateRecovery*, the pacing rate towards the target. The reaction step is determined and applied incrementally every 100 milliseconds to gradually adjust the pacing rate. A similar recovery step is used when the pacing rate needs to decrease. If the current pacing rate is less than the

target the rate, it is increased by the reaction step until the target is reached. On the other hand, if the rate exceeds the target, it is reduced using the recovery step. Finally, *adaptedRate* is returned and fed into the pacing circuit.

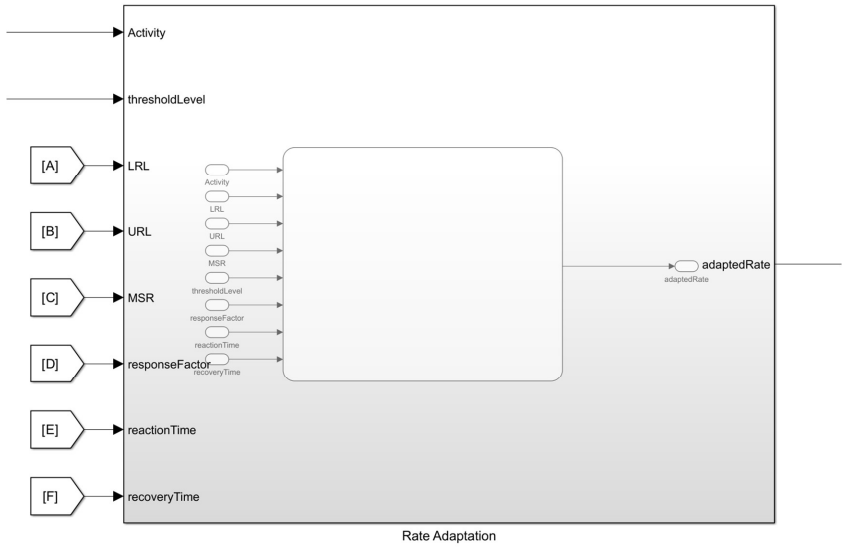


Figure 16: Rate Adaptation Subsystem

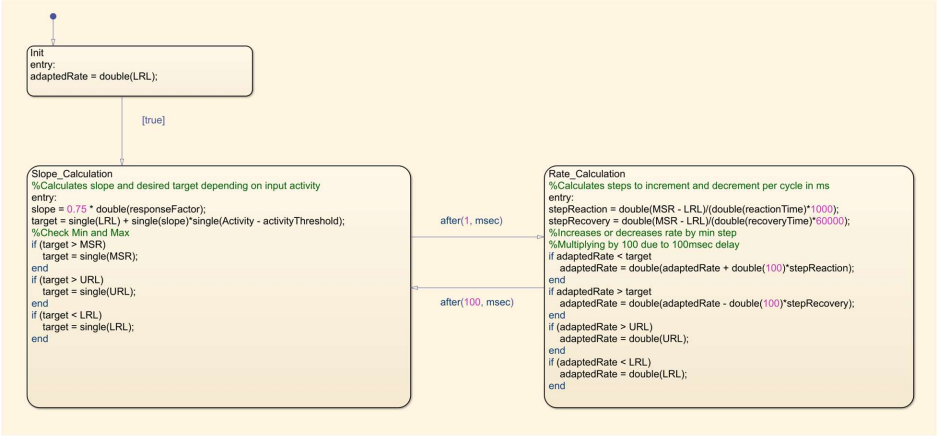


Figure 17: Stateflow Used to Calculate an Adapted Rate

Serial Communication Implementation

Serial communication is foundational for the behaviour of the pacemaker system as it allows for communication with the pacemaker. This is important for achieving some of the project requirements such as:

- Being able to adjust programmable parameters from the DCM
- Being able to verify stored parameters on the pacemaker are correct
- Graphing atrium and ventricle ECG data from the pacemaker

The most important piece of the serial communication is the establishment of the serial packet. The packet is vital as it implements safety features such as a sync byte, and the size must be managed in order to maintain a reasonable transmission rate. Our serial packet is described by the following table.

Parameter	Data Type	Bytes
Sync Byte	Uint8	0
Mode	Uint8	1
LRL	Uint8	2
URL	Uint8	3
ATR_PulseAmplitude	single	4:7
ATR_PulseWidth	single	8:11
VENT_PulseAmplitude	single	12:15
VENT_PulseWidth	single	16:19
VRP	Uint16	20:21
ARP	Uint16	22:23
MaxSensorRate	Uint8	24
ReactionTime	Uint8	25
ResponseFactor	Uint8	26
RecoveryTime	Uint8	27
ActivityThreshold	Uint8	28
AtrialData	Single[10]	29:68
VentricleData	Single[10]	69:108

Table 4: Serial Transmission Structure.

This 109-byte package consists of a sync byte followed by our programmable parameters from bytes 1 to 28. This packet is what is sent from the DCM to the pacemaker. As seen above, the packet sent from the pacemaker to the DCM also contains 40 bytes worth of each atrium and ventricle data for the purpose of graphing. This data is 10 single precision floating point numbers. This structure is constant for all transmissions in order to lower complexity while still being able to meet all requirements.

Given the 115.2 kbaud transmission rate, the pacemaker can, given the 115.2 kbaud (115 200 bits per second) transmission rate, and the fact that a full UART transmission would take 109 x 10 bits, we can send 105.69 packets per second, or every 9.4 ms. However, to allow for transmission from the DCM to the pacemaker and buffer room in case additional data would need to be sent, we chose to send data from the pacemaker to the DCM every 20 ms. This imposes constraints on the collection of graphing data as discussed below.

Because our packet calls for 10 floating point samples from each the ventricle and atrium, and we perform transition of this data every 20ms, we are constrained to perform sampling every 2ms. This is done in Simulink using the tapped delay block and sampling the analog input pins every 2ms.

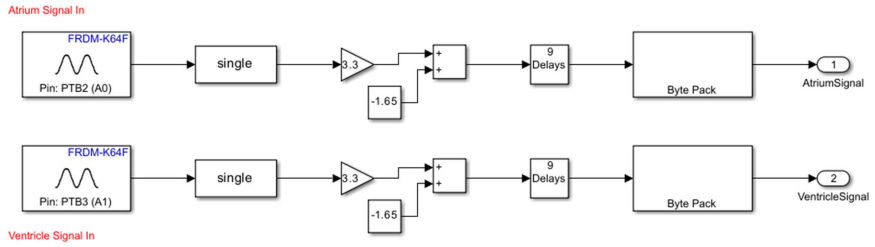


Figure 18 18: Simulink code responsible for samping atrium and ventricle data.

The addition of -1.65 was an empirically determined value arising due to a constant offset in the received data from the pacemaker.

The serial input is performed in a stateflow that receives the most recent serial transmission from the DCM and extracts the data into needed parameters.

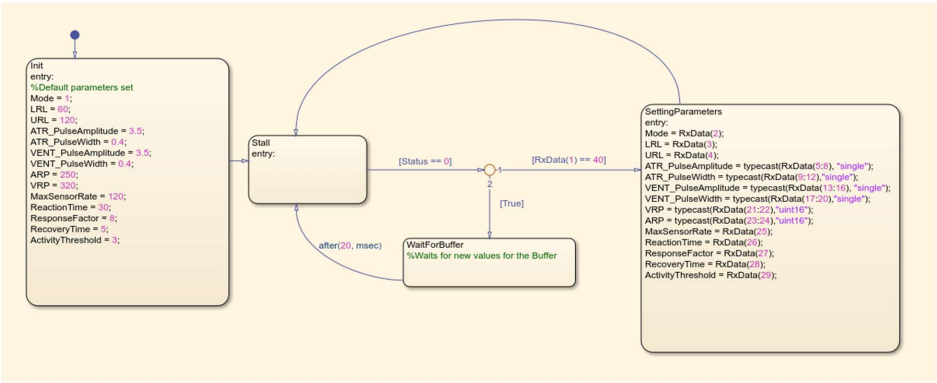


Figure 19 19: Simulink code for extracting parameters from DCM serial transmission.

The init state is responsible for setting default parameters when the code is initially flashed to the pacemaker. From there, the parameters are set via UART from the DCM. When status is equal to 0 we know that new values have been read from the input buffer. If the first byte (the sync byte) is equal to 40, the parameters are updated according to their location in the packet structure. All these received parameters are verified as you can see in the safety section of the report, as mentioned before each of the parameters are sat

The serial transmission from the pacemaker to the DCM is also vital as it allows for ECG graphing and validation of parameters.

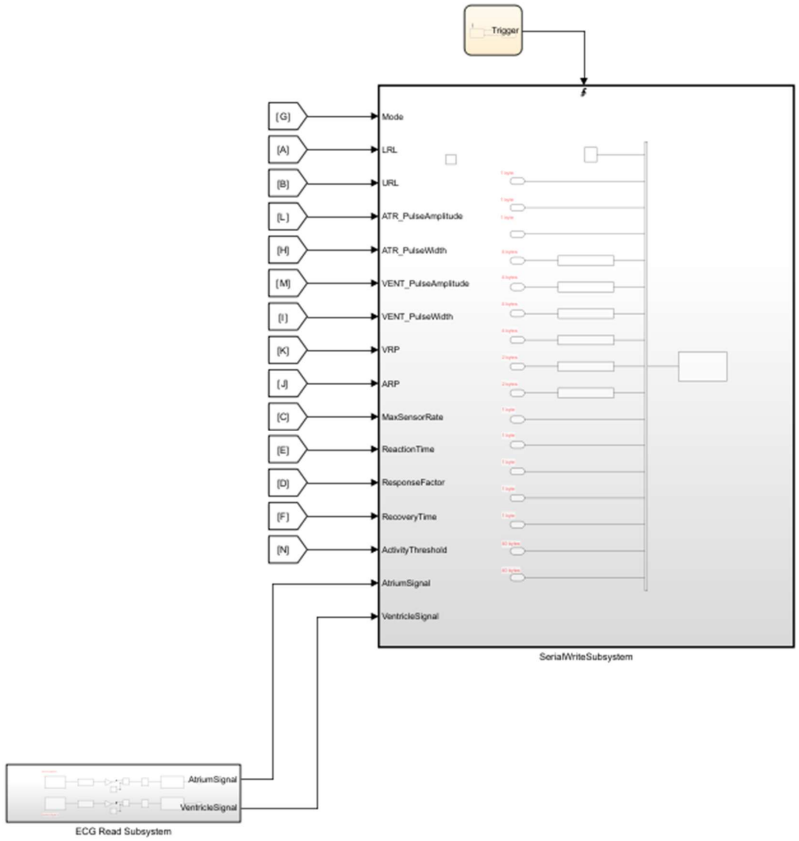


Figure 2020: Serial write subsystem and its inputs.

The subsystem takes in all the current parameters from “go to/from” blocks as shown in the above Simulink screenshot. Additionally, the ECG Read Subsystem provides a packed version of the atrium and ventricle data that will be sent for graphing as described above. The small stateflow feeding the trigger simply generates a rising edge every 20 ms in order for the subsystem to execute and serial transmission to occur at this 20 ms period.

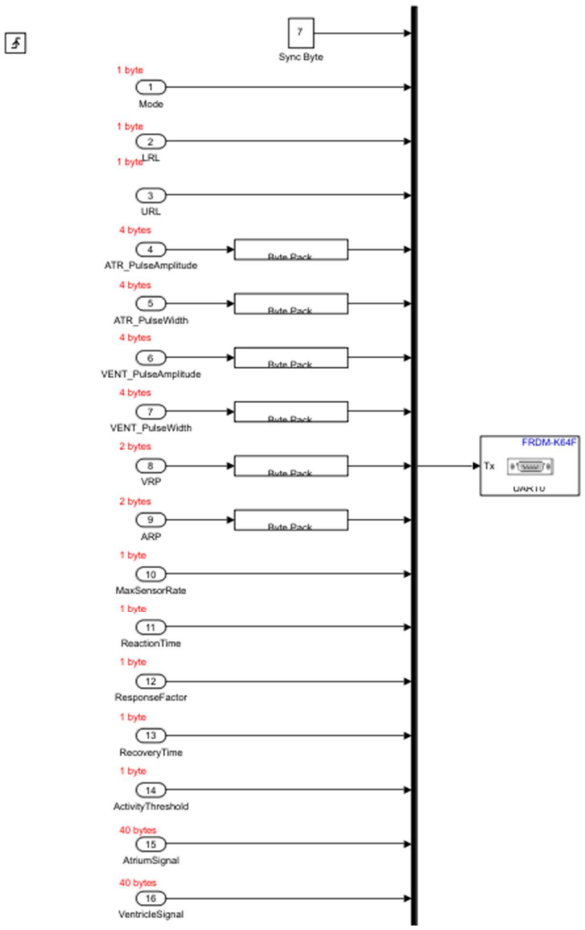


Figure 2121: Serial write subsystem.

The inner workings of the serial write subsystem are shown above. Any multi-byte signals are packed into a series of bytes. These are collapsed into a byte stream for transmission via multiplexing, before transmission over UART via the pacemaker USB port. The DCM side serial is described in another section, as is the testing of serial.

Iterative Design + Design Decisions

The pacemaker went through a multitude of development steps in order to reach its current state. The development of VOO and AOO was tackled first in order to confirm the functionality of the

Commented [TS9]: Add more examples for assignment 2. Can talk about evolution of serial structure and changing from a bunch of different message types to 1, etc.

Commented [TS10R9]: Add example with serial

charging and discharging behaviour. While VOO was quick to come together, AOO was slow going as many hours of debugging were disregarded when it was revealed one hardware kit is faulty as the same AOO code worked properly on another hardware kit.

From there, work began on AAI and VVI. The main hurdle in putting together the sensing logic was determining whether to use 1x or 100x sensing attenuation. 100x was empirically determined as the correct choice.

It was important to maintain a high degree of modularity in the code, striving for low coupling between modules. This is why the top-level states are arranged in operating states rather than subtasks such as pacing and charging. This allows only one input to be required (the mode), while the substates take care of their own computations. As only one state is active at a time, this is not a computational burden.

In beginning the second assignment, a great deal of design thought went into the serial communication as described in the *Serial Communication Implementation* section. A first major decision was to use the smallest data type that could store a given programmable parameter in all cases. For floating point values, a single is used. All other values can be represented by an integer and so utilize either uint8 or uint16 data type depending on the maximum size. Additionally it was decided to transmit every 20ms as discussed. This drove the decision to sample atrium and ventricle data every 2ms.

When tackling rate adaptive pacing, there were a number of decisions made in order to achieve functionality akin to that described in the *rate_adaptive_pacing* document. After observing the raw acceleration magnitude data, it was chosen to scale it by a gain of 40 to achieve the desired activity level. Different activity thresholds were chosen to correlate to different activities empirically by simulating various levels of activity. The design process for rate adaptive pacing is described in the section *Rate Adaptive Pacing Implementation*.

Safety

Commented [TS11]: Should probably expand this

This is a safety critical system. A major consideration was to ensure that the atrium or ventricle is never connected to the charging capacitor while it was charging, resulting in a direct connection to the charging PWM signal. Additionally, as we look into the future and the connection of the DCM to the pacemaker it will be vital that we perform checks to confirm the validity of parameters received from the DCM. The most important thing we can do to ensure that the system is safe isNon to ensure that our modes work properly, as is validated in the *Testing - Pacemaker* section.

The entire application of the FRDM pacemaker hinges on the notion that system is designed to highest of safety standards. Much like the construction assignment 1 previous safety features were carried over, in other words the charging PWM signal is not directly connected to the user's ventricle, and atrium. Like advised in the interview discussion caps/limits have been employed on both sides, when sending and receiving from the DCM. In the case of Simulink's implementation, the inputs have been run through "saturation blocks" which set upper and lower limits, meaning if a value is lower than the lower limit the value is clipped to the lower limit, as per a case for a higher value that upper limit something similar happens.

This acts as a layer of redundancy as the proper parameter values are also checked on the DCM side. This means that both the pacemaker and the DCM perform value capping. The redundant test on the pacemaker side ensures that no erroneous state will be entered if the pacemaker clipping has a bug, or if a communication error occurs.

An additional safety feature for the system is the implementation of parameter validation on the DCM. When parameters are echoed back to the DCM from the pacemaker, they are checked for matching the last sent parameters. If they do not match, the user is notified that there is a potential error with the system.

Modules – DCM

Mode (Backend)

Purpose

The Mode class defines several operational parameters that determine the pacemaker's behavior. The class has four methods, two of which are setters and two of which are getters.

Functions

Function Name	Encapsulation	Arguments (type)	Black Box Description	Return Type
set_name	Public	String	Set the name of the mode.	None
set_parameters	Public	None	Set the Pacemaker's parameters, depending on the mode.	None
get_mode	Public	None	Retrieve the mode's name.	String
get_params	public	None	Retrieve the parameters depending on the mode.	Strings

Internal Behaviours

- set_name: the method sets the name of the Pacemaker's operational mode. For example, the user can set the mode to VVI.
- set_parameters: the method allows the user to set up to thirteen Pacemaker parameters, depending on the mode. The parameters are lower rate limit, upper rate limit, atrial amplitude, atrial pulse width, ventricular amplitude, ventricular pulse width, atrial refractory period, ventricular refractory period, reaction time, recovery time, response factor, activity threshold, and maximum sensor rate. However, the user will not have to set all parameters if the mode does not call for it.
- get_mode: the method returns the mode name.
- get_params: the method returns the thirteen Pacemaker parameters, even if they're set to Null.

User (Backend)

Purpose

The User class (module) is the skeleton for fetching new user data by providing the required username or password. It makes the code object-oriented rather than retrieving data solely from CTK entries.

Functions

Function Name	Encapsulation	Arguments (type)	Black Box Description	Return Type
get_username	Public	None	Retrieve the user's username.	String
get_password	Public	None	Retrieve the user's passowrd	String

Internal Behaviours

- get_username: the method works by returning the user's username as a string.
- get_password: the method works by returning the user's password as a string.

Window (Backend)

Purpose

The Window class is decomposable into two backend components: signing in (including registering and changing the password) and creating the electrogram charts. The signing-in component consists of three submodules: registering the user, signing in to the DCM, and changing the password. Registering the user involves providing a username and password, then clicking the register button, and signing in requires the user to input their credentials to log in. After signing into the DCM, a user can change their password. The electrogram charts are displayed immediately upon entering the main Pacemaker page.

Functions

Table 5: Module Backend Methods

Function Name	Encapsulation	Arguments (type)	Black Box Description	Return Type
handle_signin	Public	None	If the entered username and password matches the one in the text file, the Pacemaker page opens. Otherwise, the	None

			program notifies the user their credentials don't work.	
handle_register	Public	None	If the entered username doesn't exist and there are less than 10 users, the program writes the new credentials to the file.	None
hash_password	Public	password (string)	Takes the entered password from the sign-in page and hashes it.	String
enter_info	Public	username (string), newpassword (string), label (tk.label)	If the entered username matches the one in the file, the newly entered password is hashed and written to the file as the user's primary password.	None
ventricular_ electrogram	Public	None	Creates a matplotlib figure and attaches it to a TK canvas.	CTK Canvas
atrium_ electrogram	Public	None	Creates a matplotlib figure and attaches it to a TK canvas.	CTK Canvas
save_paramaters	Public	None	Sends parameters entered in CTK entry boxes to .json file.	None
verify_params_json	Public	None	Verifies Pacemaker data matches the. json values.	None
verify_params	Public	None	Check if the. json parameters match the Pacemaker's parameters every serial transmission.	None
activity_thresh_converter	Public	String	Converts the activity threshold string into an integer index	Int
check_serial	Public	None	Check serial connections and verify .json values (with a 20 ms delay)	None

connect_pm	Public	None	Attempt to connect the Pacemaker (depends on user's laptop and port number)	None
isConnected	Public	None	Returns the connection status, and the port number if properly connected.	String
update_connection_status	Public	None	Updates the connection status CTK label	None
disconnect_pm	Public	None	Attempts to disconnect the Pacemaker from the specified port.	None

Internal Behaviours

- handle_signin: the method works by retrieving the user input from the TKinter entry widgets and then invoking the hash password method to encrypt the password. Subsequently, the method reads the local file and splits all the lines into a list of strings. The process iterates over each line and splits the lines into another list of strings. If the entered username matches the first index of the split string and the hashed password matches the second index, the Pacemaker page opens. Otherwise, if the wrong credentials are received, the method exits. If the loop completes without finding a matching username, the user is notified (Figure 22).

```python
# Function for processing sign in of user (already registered)
1 usage  fostej26 +2
def handle_signin(self):

    # Retrieve the username and passwords from the tk_entries
    username = self.username_entry.get()
    password = self.password_entry.get()
    hashed_password = self.hash_password(password)

    # Open the text file
    with open("users.txt", "r") as f:

        # Separate the user list in the file
        users = f.read().splitlines()
        for user in users:
            user_data = user.split()
            if username == user_data[0]:  # Check if the provided username matches the first entry in the line
                if hashed_password == user_data[1]:  # Check if the hashed password matches the second entry
                    # in the line
                    self.message_label.configure(text="Sign In successful!", text_color="green")
                    self.init_pacemaker_page()  # Open pacemaker page
                    return
                else:
                    # Message displayed when password is incorrectly inputted
                    self.message_label.configure(text="Incorrect password. Please try again.", text_color="red")
                    return
    self.message_label.configure(text="Username not found. Please register.")
```

Figure 22 22: handle_signin function

- handle_registration: the handle_register method manages the user registration process. It starts by retrieving the user input from the Tkinter entry widget and opening the user data file. The method splits the lines into strings and checks the user count. If the number of users exceeds ten, the label notifies the user, and the process exits. If the username exists, the label notifies the user of an existing profile and exits. Otherwise, the method appends the new credentials to the file, clears the entries, and exits.
- hash_password: the method takes the password from the Tkinter entry and uses the SHA-256 hashing function to protect the credentials.
- enter_info: the handle-change_password method changes the user's password. It starts by checking empty entries and exiting if the conditions are correct. Next, the process hashes the new password and opens the file to split the lines. The method initializes a flag to track whether the username has been found. Furthermore, the process iterates through each line and matches the username to the first index of the newly split string. If a username matches an index, the method updates the line, sets the flag to become true, and exits the for loop. If the flag is still false, the process notifies the user and returns. The file finally opens in write mode and appends the updated list of users to the file.

```
213  class Window(ctk.CTk):
541      def enter_info(self, username, newpassword, label):
542
543          # Check if the entries are empty
544          if username == 0 or newpassword == 0:
545              label.configure(text="No empty entries please")
546              return
547
548          # Hash the newly entered password
549          new_hashed_password = self.hash_password(newpassword)
550
551          try:
552              # Open the text file with user data
553              with open("users.txt", "r") as f:
554                  users = f.read().splitlines()
555
556              # Initialize a flag for if the username is found in the file
557              found_flag = False
558
559              # Use the enumerate function to iterate through the indexed user file
560              for i, user in enumerate(users):
561                  user_data = user.split()
562
563                  # If the entered username matches the username at the index, add the newly hashed password
564                  if username == user_data[0]:
565                      users[i] = f"{username} {new_hashed_password}"
566
567                      # Set the flag and break out of the loop
568                      found_flag = True
569                      break
```

Figure 2323: enter_info initialization

- ventricular_electrogram: the method creates an animated plot of a ventricular electrogram. The process starts by initializing a figure and axes for the plot with a specified size alongside the axis's titles. Next, the method initializes the line and sets the plot limits. The animation function updates the plot in every frame with the given equation. Finally, the process creates the animation using the matplotlib library and embeds the figure in a Tkinter frame., while returning the canvas.
- atrium_electrogram: see ventricular_electrogram.
- save_parameters: the method captures, validates, and prepares the Pacemaker parameters to be sent based on the user's input and selected mode. Firstly, it retrieves all the user-entered data and pads the entries with default values if any input is missing. Next, the method invokes the checkparams function to verify the values and check the mode index. Using a byte array (struct.pack), the process encodes the Pacemaker data and sends it via the send_data function. It also creates a dictionary (mode_data) for future reference and assigns default values if any entries are left empty, ensuring the system updates with safe settings.
- activity_thresh_converter: the method takes a string as input and converts it to an integer index, used by other methods within the class. For example, if the user selects "H" (high), the corresponding value is 5.
- verify_params_json: the method first tries to open the parameters.json file and save the data under a specified user. If that fails, the statement throws an exception. Next, the method obtains the mode via a getter and matches the information to the DCM.

- verify_params: the method invokes the check_params_json procedure and calls it after 20 ms.
- check_serial : the method grabs the username and current mode and then checks the active serial port. The Main module invokes the method after 20 ms.
- connect_pm: the method tries to connect to the serial port and updates the connection status label text. It also checks the status after 20 ms. However, in the event of an unsuccessful connection, the method throws an exception and updates the status label.
- is_connected: the method returns a different string depending on the connection_status state. If it's true, it'll return the Pacemaker's port. Otherwise, it'll prompt the user to check their device or computer.
- update_connection_status: the method updates the status_text variable depending on the result of the isConnected method. Subsequently, it updates the connection status label.
- disconnect_pm: if there's a device connected to the computer, the method will close the port and output a successful disconnection message. Otherwise, it will inform the user there's no device to disconnect. Both directions update the connection status label.

Window (Frontend)

Purpose

The frontend of the DCM follows a 3-page design, with the pages being: Login, Pacemaker, and Change Password. The pages are compiled and transitioned between via the use of 6 methods:

Functions

Function Name	Encapsulation	Arguments (type)	Black Box Description	Return Type
Init	Public	None	Initialize common window components such as window size and name. Calls upon Init_Login_Page to begin stateflow	None
Init_Login_Page	Public	None	Compiles the logo, username and password textboxes, and sign-in/register buttons. Buttons call upon handle methods.	None
Init_Pacemaker_Page	Public	None	Compiles navbar, pacemaker connection status, pacemaker modes, and electrograms. Contains buttons that direct to Login	None

			and Change password pages.	
Change_Password_Page	Public	None	Clears previous window. Compiles the logo, username and password textboxes, and change password button and return button. Buttons call upon handle methods.	None
Show_Pacemaker_Page	Public	None	Deletes current window and calls upon Init_Pacemaker_Page	None
Show_Login_Page	Public	None	Deletes current window and calls upon Init_Login_Page	None
init_AOO	Public	None	Deletes the current parameters frame and initializes the AOO parameters frame.	None
init_VOO	Public	None	Deletes the current parameters frame and initializes the VOO parameters frame.	None
init_AAI	Public	None	Deletes the current parameters frame and initializes the AAI parameters frame.	None
init_VVI	Public	None	Deletes the current parameters frame and initializes the VVI parameters frame.	None
init_AOOR	Public	None	Deletes the current parameters frame and initializes the AOOR parameters frame.	None
init_VOOR	Public	None	Deletes the current parameters frame and initializes the VOOR parameters frame.	None

init_AAIR	Public	None	Deletes the current parameters frame and initializes the AAIR parameters frame.	None
Init_VVIR	Public	None	Deletes the current parameters frame and initializes the VVIR parameters frame.	None
toggle_ventricular_graph	Public	None	ISR for enabling/disabling the ventricular electrogram within the electrogram frame.	None
mode_selected	Private	*args	A helper method that disables the save parameters button if a mode has not been selected.	None
toggle_atrium_graph	Public	None	ISR for enabling/disabling the atrial electrogram within the electrogram frame.	None

Serial Transmission (Backend)

Purpose

The serial transmission module has several purposes. It manages serial communication by connecting and configuring the port and sending data. It also parses data and verifies the received data with the .json file. The module stores data with the associated username and graphs it on the CTK page. Finally, it decodes and organizes the parameters via the Params class.

Functions

Function Name	**Encapsulation**	**Arguments (type)**	**Black Box Description**	**Return Type**
connect_serial_port	Public	None	Connects to a serial port that is open. Raises an exception if	None or Exception

			port does not open.	
connection_status	Public	None	Checks if the serial port is open	Boolean
get_port	Public	None	Returns the port name	String
close_serial	Public	None	Closes the serial port	None
get_current_serial_counter	Public	None	Returns the value of the serial counter	Integer
get_atr_vent_graphing_data	Public	None	Returns the atrial and ventricular graphing data for visualization	List, List
send_data	Public	data : bytes	Sends data over serial port if open	None
verify_data	Public	current_username : string, current_mode : string, data : dictionary, filename : string	Verifies incoming data for stored parameters in Json file	Boolean
check_serial_port	Public	username : string, mode : string	Read data from serial, parses, and updates graphing data	None

Internal Behaviours

- connect_serial_port: the method works by setting the serial ports baud rate, name and timeout. Then it will try to open the serial port. If it fails, a exception will be raised with error message. If serial port is opened successfully, the port is left open.
- Connection_status: the method is a getter that works by checking if the global port object is open, this function then returns its status
- Get_port: this method is a getter, that works by getting the serial ports name, by accessing the global port object, and getting it's name.
- Close_serial: this method works by closing the serial port by using the close method on the global port object, then it resets the global counter variable

- Get_current_serial_counter: this getter method access the value of the global counter variable.
- Get_atr_vent_graphing_data: this method gets the global atrium graphing data and ventricle graphing data.
- Send_data: this method starts by checking if serial port is open, then if open, converts data to bytes and sends it to the port, if the port is not open an error message is raised
- Verify_data: This method works by reading the json file with the stored user parameters, it then checks the data for the required current username and mode. Then compares the received data with the stored user parameters. Integer values are ranged with a tolerance of 5%, and if any key is missing or has incorrect values, an error message is raised.
- Check_serial_port: this method begins by check if the port is open, then it read the first byte to check that it is at the start indicator. Then read the next 108 bytes into byte array and parses into structured parameters. The parsed data is then verified, then the atrial and ventricular graphing data from the byte array is extracted. Then appended to each global lists, both lists are trimmed to retain only the 5000 values at a time.

Design and Considerations

- Login Page:

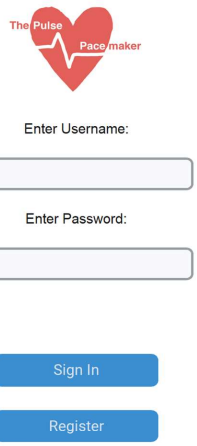


Figure 24: Login Page

The login page allows the user to enter their username and password and either sign in to their existing account or create a new one. In the event an error occurs, the message will be displayed above the sign in button. These error messages provide insight on why they were unable to login/register.

- Pacemaker Page:

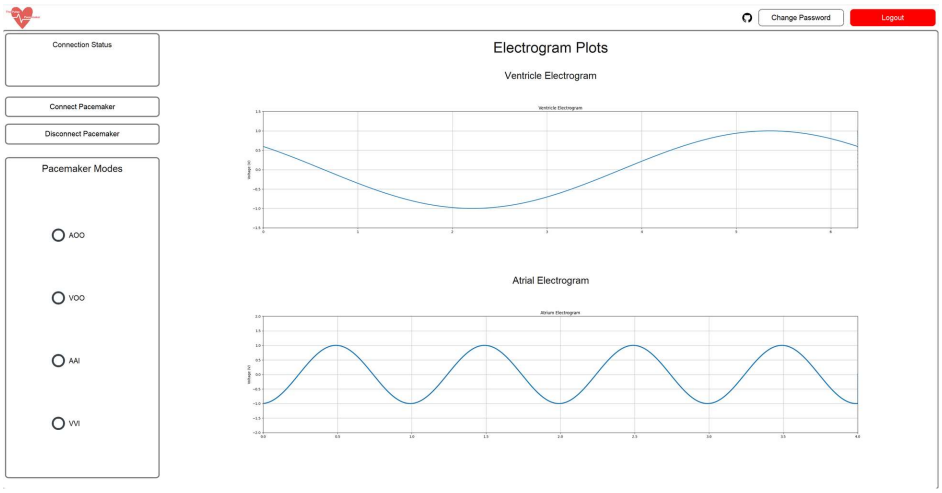


Figure 25: Initial Pacemaker Page

The pacemaker page allows users to connect/disconnect their pacemaker, alter any parameters, and view both atrial and ventricular electrograms. Additionally, they are also able to log out and return to the sign-in page, change their password, or view the source code. A save button was added to the parameters to ensure that the correct values are applied to the pacemaker for safety reasons.

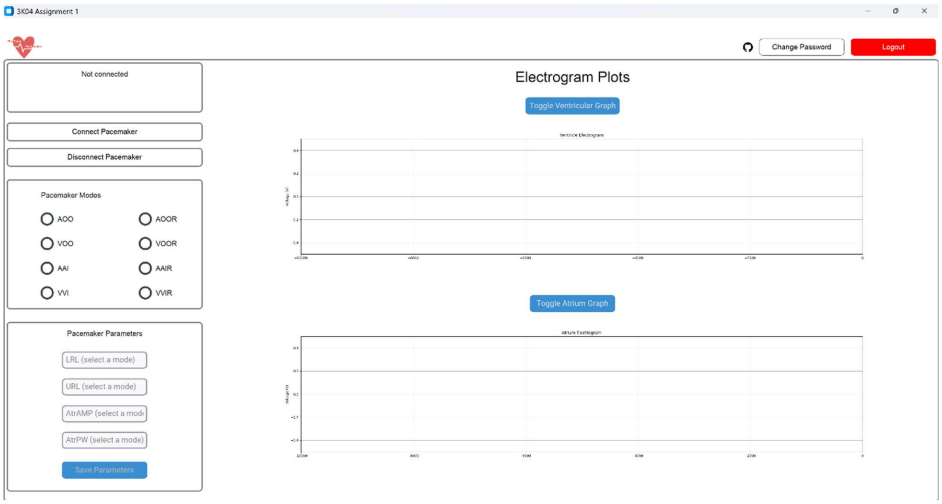


Figure 26: Final Pacemaker Page

Upon the addition of new modes and parameter requirements, the pacemaker page was significantly altered to conform to new requirements and maintain user experience. A frame containing input boxes for the pacemaker parameters was added, as well as the addition of new modes to the existing modes frame. For the purposes of safety and clarity, all valid input ranges have been outlined as the placeholder text inside each input box. If the user does not comply with these requirements, they will be prompted with a message notifying them that at least one of their inputs is invalid. Additionally, the ability to toggle the display of individual electrograms was also added to allow the user to disable an irrelevant graph (i.e. the ventricular graph in AOO mode).

- Change Password Page:

Figure 27: Change Password Page

The login page allows the user to enter their current username and a new password. In the event an error occurs, the message will be displayed above the change button. These error messages provide insight on why they were unable to login/register. After a successful password change, a message will inform the user that their password has been reset and prompt them to return to the pacemaker screen.

Testing - DCM

Commented [TS12]: Needs significant expansion

The login functionality for the DCM was tested by iterating through the following state flow:

$$Register \rightarrow Log\ In \rightarrow Pacemaker\ Page \rightarrow Change\ Password \rightarrow Pacemaker\ Page \rightarrow Log\ Out \rightarrow Repeat$$

This state flow was repeated until the maximum of 10 users was reached and was passed successfully. Upon reaching the 10-user mark, the user is prompted with a message notifying them that the maximum number of users on the device has been reached.

Once logged in, the user has the option to connect a pacemaker to the DCM (Figure 28). If the user selects the “Connect pacemaker” button when a pacemaker is not connected, the status frame will notify the user that an attempt to connect to a device was made (Figure 29), but no devices were found connected to the computer.

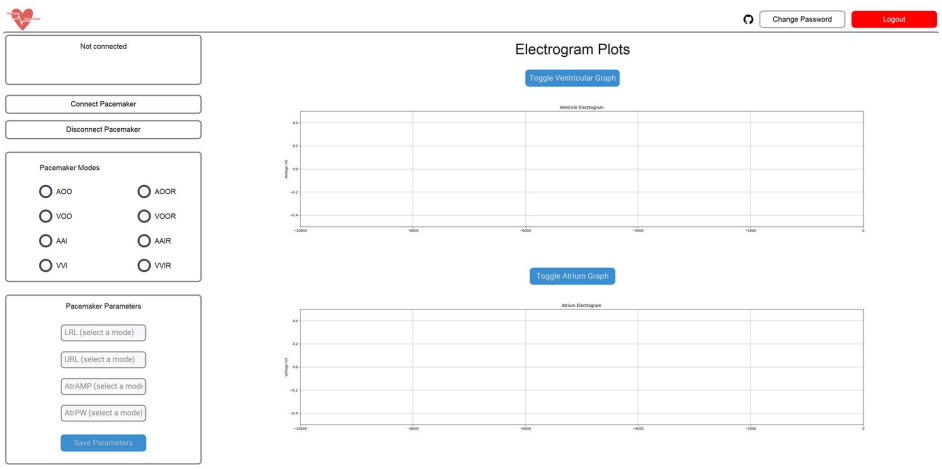


Figure 28: Pacemaker disconnected

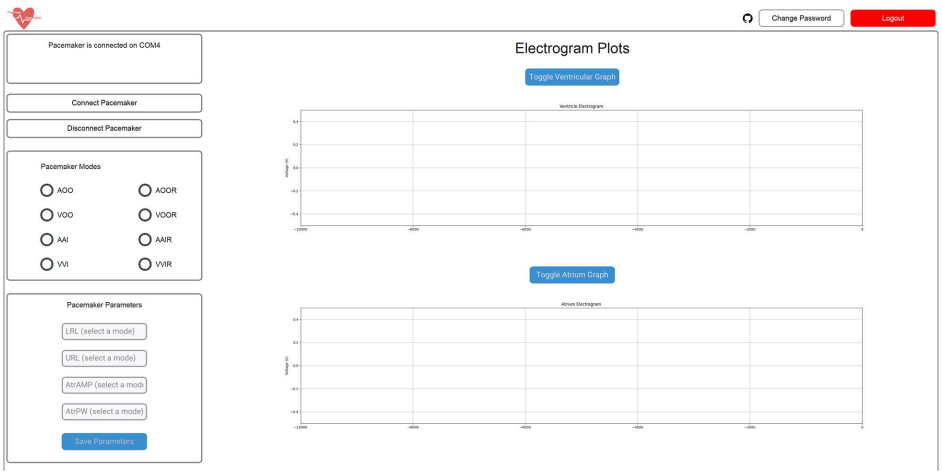


Figure 29: Pacemaker connected

Once a pacemaker has been connected, the user is able to select a mode, however if a mode has not been selected or a connection to a pacemaker has not been found, the user is unable to submit their inputs to the DCM. If a pacemaker is found to be connected, the user can submit parameters for the mode they selected. Each parameter input box contains a range of acceptable input values (Figure 30). If the values are outside of that range, the inputs will be refused, and the user will be notified.

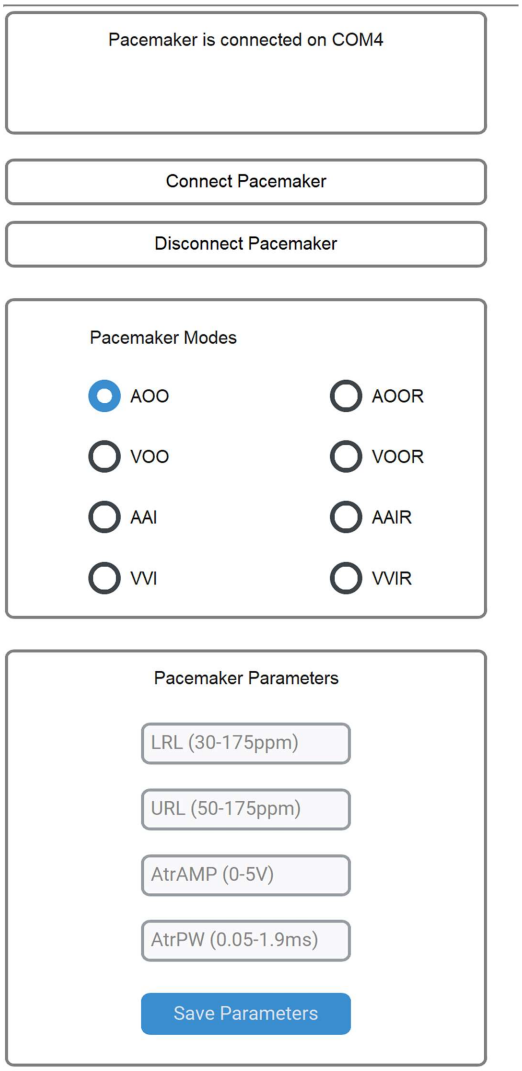


Figure 30: Connected Pacemaker + AOO mode parameters and ranges

Once the user has sent values to the pacemaker, the DCM will send the parameters to the pacemaker and overwrite the existing ones. To prevent malfunctions with consequences on patients' health, the DCM retrieves and validates the parameters from the pacemaker to ensure they match the inputs that were submitted. If these inputs do not match, the pacemaker is automatically disconnected from the DCM.

Parameters – DCM

Once the user enters parameters into the DCM, they are written into a JSON file as a 3-dimensional array. The outer list is a list of users, the middle list is a list of modes for that user, and innermost list is a list of parameters for that mode.

At the same time the JSON file is written, a UART transmission containing all parameters in the form of a 138-byte array is made, sending the parameters to the hardware for execution.

When the pacemaker is connected and begins serial transmission, the DCM retrieves the parameters currently operating on the pacemaker and compares them to the JSON file values that were inputted. For any numerical values, the feedback values are considered valid if they fall within a 0.5% window of relative error.

All parameters are saved as strings to the file and transmitted as integer and float values. Type transfers occur directly before serial transmission. The design utilizes string inputs since some parameters are not numerical values and are unable to be compared using a relative error calculation.

Testing – Pacemaker

AOO

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude (V)
AOO	60	0.4	3

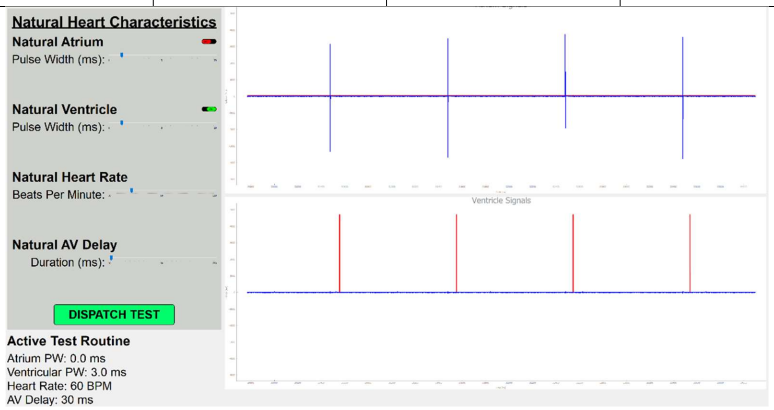


Figure 31: AOO 60 BPM Test.

In this test the desired BPM is set at 60. This causes the atrium to be paced by the pacemaker at a constant rate corresponding to 60BPM. This pacing correctly occurs regardless of any other heart activity, such as the ventricle activity seen in the lower half of the figure. The ventricle BPM is set to 60BPM to show the correct pacing rate of the atrium.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude (V)
AOO	80	0.6	4V

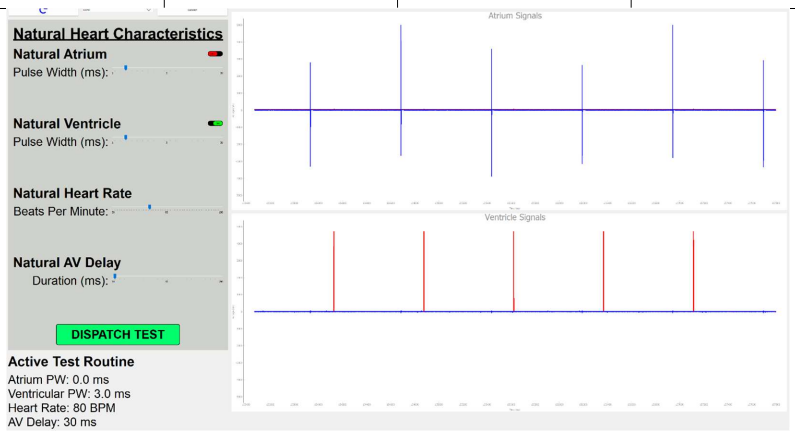


Figure 32: AOO 80 BPM Test.

In this test the desired BPM is set at 80. This causes the atrium to be paced by the pacemaker at a constant rate corresponding to 80BPM. This pacing correctly occurs regardless of any other heart activity, such as the ventricle activity seen in the lower half of the figure. The ventricle BPM is set to 80BPM to show the correct pacing rate of the atrium.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude (V)
AOO	60	0.4	3V

Figure 33: AOO Test with other atrium signals.

In this test the desired BPM is set at 60. This causes the atrium to be paced by the pacemaker at a constant rate corresponding to 60BPM. This pacing correctly occurs regardless of any other heart activity, as highlighted by the fact that the correct pacing is maintained from the pacemaker (blue pulses) despite the 66 BPM natural pacing of the heart.

VOO

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude (V)
VOO	60	0.4	3

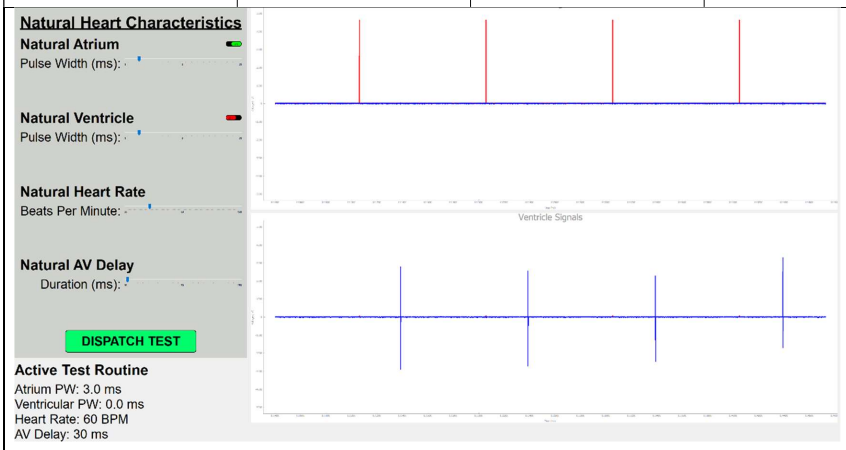


Figure 34: VOO Test 60BPM.

In this test the desired BPM is set at 60. This causes the ventricle to be paced by the pacemaker at a constant rate corresponding to 60BPM. This pacing correctly occurs regardless of any other heart activity, such as the atrium activity seen in the upper half of the figure. The atrium BPM is set to 60BPM to show the correct pacing rate of the atrium.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude (V)
VOO	80	0.6	4

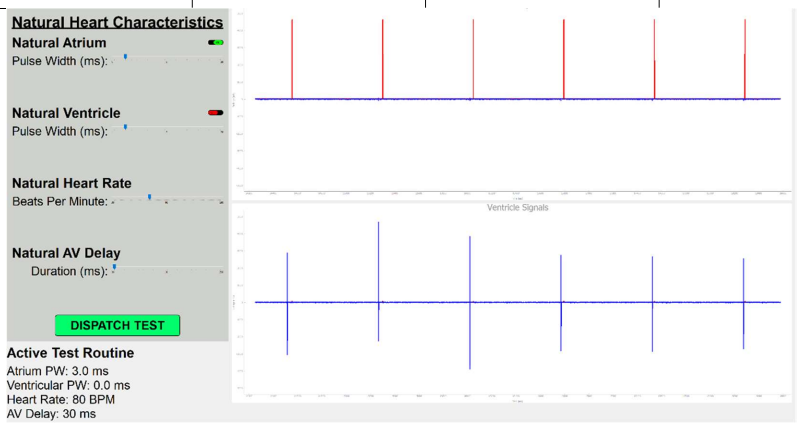


Figure 35: VOO Test 80 BPM.

In this test the desired BPM is set at 60. This causes the ventricle to be paced by the pacemaker at a constant rate corresponding to 60BPM. This pacing correctly occurs regardless of any other heart activity, such as the atrium activity seen in the upper half of the figure. The atrium BPM is set to 60BPM to show the correct pacing rate of the atrium.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude (V)
VOO	60	0.4	3V

Figure 36: VOO Test with other ventricle signals.

In this test the desired BPM is set at 60. This causes the ventricle to be paced by the pacemaker at a constant rate corresponding to 60BPM. This pacing correctly occurs regardless of any other heart activity, as highlighted by the fact that the correct pacing is maintained from the pacemaker (blue pulses) despite the 66 BPM natural pacing of the heart.

AAI

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
AAI	40	0.6	3	100

Figure 37: AAI Test at 40BPM setting.

In this test the desired BPM is set to 40. However, because the natural heart rate is continuously beating at 60BPM, the output pulses are continuously inhibited. This means that the pacemaker never provides any pacing as none is required due to the natural heart rate being greater than the desired pacemaker pacing rate.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
AAI	60	0.5	3	100

Figure 38: AAI Test at 60 BPM setting.

In this test the desired BPM is set to 60. However, because the natural heart rate is continuously beating at 60BPM, the output pulses are continuously inhibited. This means that the pacemaker never provides any pacing as none is required due to the natural heart rate being equal to the desired pacemaker pacing rate.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
AAI	80	0.5	3	100

Figure 39: AAI Test 80 BPM setting.

In this test the desired BPM is set to 80, and the natural heart is beating at 60 BPM. 80BPM pacing requires a pulse every 750ms. Using the scale at the bottom of the atrium chart (where each numerical marking denotes a second), it is clear that the heart is being paced by the pacemaker following a period of inactivity totalling 750ms after a natural pulse as desired.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
AAI	65	0.5	3	100

Figure 40: AAI Refractory Period Test.

In this test the desired BPM is set to 65, and the natural heart is beating at 60 BPM. 65BPM pacing requires a pulse every 923ms. It is clear that there is an alternating pattern in gaps between given blue and red pulses. This occurs due to the refractory period. When the blue and red pulse are further apart, the red pulse inhibits the pacemaker, so the next blue pacemaker pulse occurs 923ms later. However, then the natural heart beats 77ms after that blue pulse. Because the refractory period is 100ms, this inhibition is ignored and the next blue pulse comes 923ms after the last blue pacemaker pulse. This repeats continuously, matching the desired refractory period design.

VVI

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
VVI	40	0.6	3	100

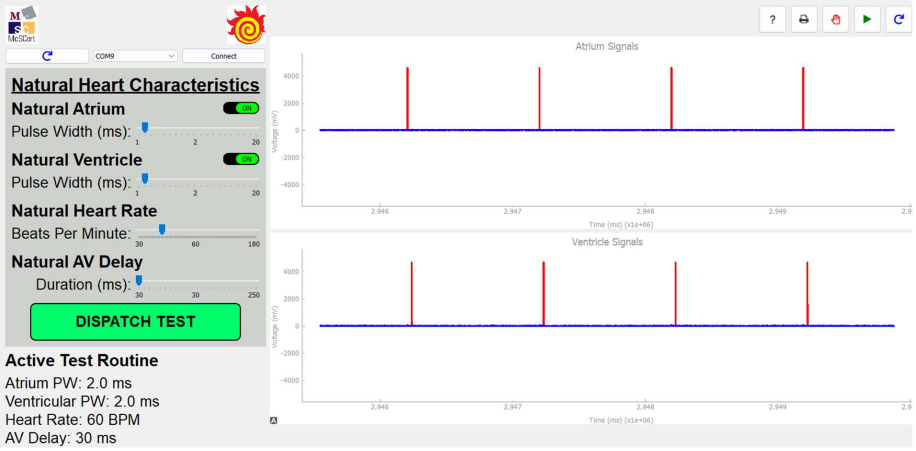


Figure 41: VVI Test 40 BPM setting.

In this test the desired BPM is set to 40. However, because the natural heart rate is continuously beating at 60BPM, the output pulses are continuously inhibited. This means that the pacemaker never provides any pacing as none is required due to the natural heart rate being greater than the desired pacemaker pacing rate.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
VVI	60	0.5	3	100

Figure 42: VVI Test 60BPM setting.

In this test the desired BPM is set to 60. However, because the natural heart rate is continuously beating at 60BPM, the output pulses are continuously inhibited. This means that the pacemaker never provides any pacing as none is required due to the natural heart rate being equal to the desired pacemaker pacing rate.

Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
VVI	80	0.5	3	100

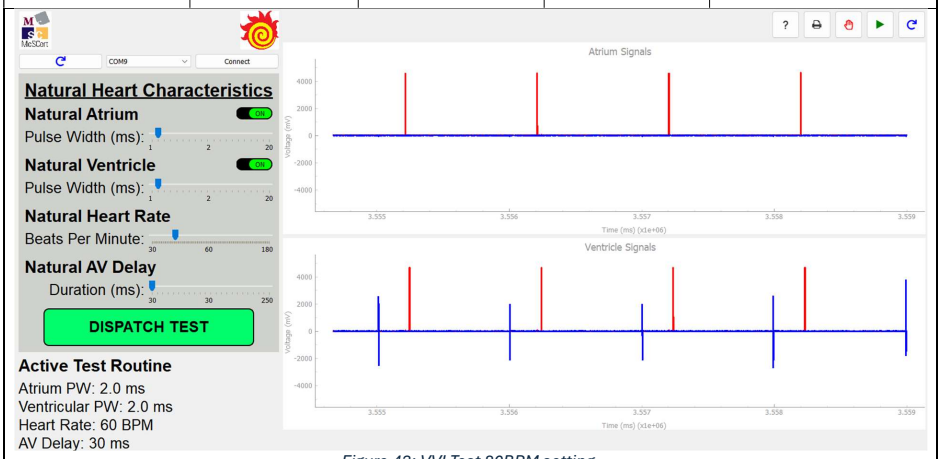


Figure 43: VVI Test 80BPM setting.

In this test the desired BPM is set to 80, and the natural heart is beating at 60 BPM. 80BPM pacing requires a pulse every 750ms. Using the scale at the bottom of the ventricle chart (where each numerical marking denotes a second), it is clear that the heart is being paced by the pacemaker following a period of inactivity totalling 750ms after a natural pulse as desired.

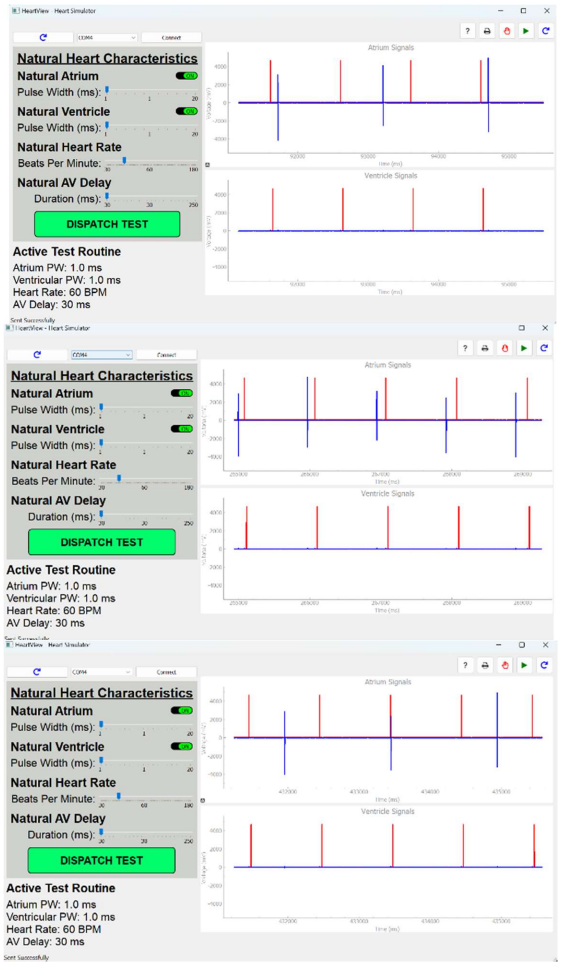
Mode	LRL (BPM)	Pulse Width (ms)	Pulse Amplitude	Refractory Period (ms)
VVI	65	0.5	3	100

Figure 44: VVI Refractory Period Test.

In this test the desired BPM is set to 65, and the natural heart is beating at 60 BPM. 65BPM pacing requires a pulse every 923ms. There is an alternating pattern in gaps between given blue and red pulses. This occurs due to the refractory period. When the blue and red pulse are further apart, the red pulse inhibits the pacemaker, so the next blue pacemaker pulse occurs 923ms later. However, then the natural heart beats 77ms after that blue pulse. Because the refractory period is 100ms, this inhibition is ignored, and the next blue pulse comes 923ms after the last blue pacemaker pulse. This repeats continuously, matching the desired refractory period design.

AOOR

Mode:	AOOR
LRL:	40 BPM
URL:	65 BPM
MSR:	65 BPM
Atrial Pulse Amplitude:	3 V
Atrial Pulse Width:	1.0 ms
Activity Threshold:	Low
Response Factor:	4
Reaction Time:	15 sec
Recovery Time:	2 min

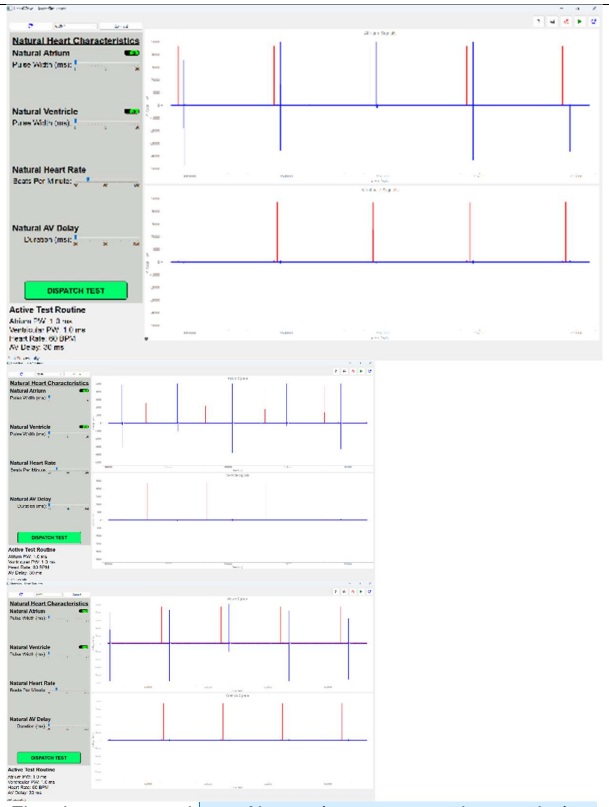


The above screenshots of heart view represent the pre, during, and post physical activity response respectively. As shown in the screenshots the natural BPM is set to 60. During the shake the pulses sent to the atrium paces regardless of current atrium activity at a higher rate due to the exerted activity. The screenshot showcases a realistic atrium pulse after the physical activity as bpm is more clearly increased to an upper limit of 65 BPM.

Commented [AN13]: AOOR doesn't do sensing.

Commented [AN14]: the response factor doesn't act like a pulse delay its like a sensitivity measure for the pace maker

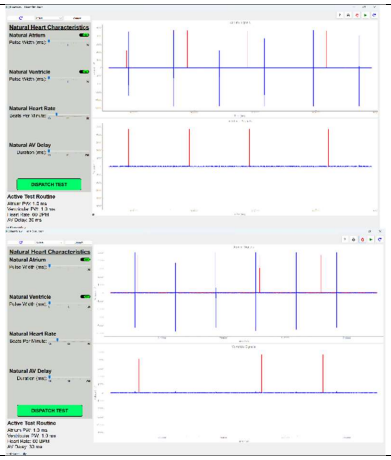
Mode:	AOOR
LRL:	60 BPM
URL:	80 BPM
MSR:	80 BPM
Atrial Pulse Amplitude:	4.2 V
Atrial Pulse Width:	0.9 ms
Activity Threshold:	Med-High
Response Factor:	11
Reaction Time:	10 sec
Recovery Time:	2 min

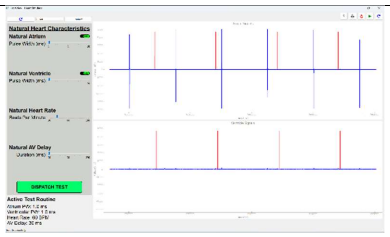


The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Although the threshold to activity is higher than previous the response factor is raised to a significant degree with provides the second screenshot an appropriate raise to 80BPM. The first sign of reaction to the physical activity was around 467000ms. The agitation started

		around 453000ms this roughly matches the reaction time of 10 seconds as the difference was 14000ms. Proceeding this the pacemaker naturally reverts to the pre – shake state, this was after a period of 488000ms to 610000ms, which reflects a 2-minute recovery time.

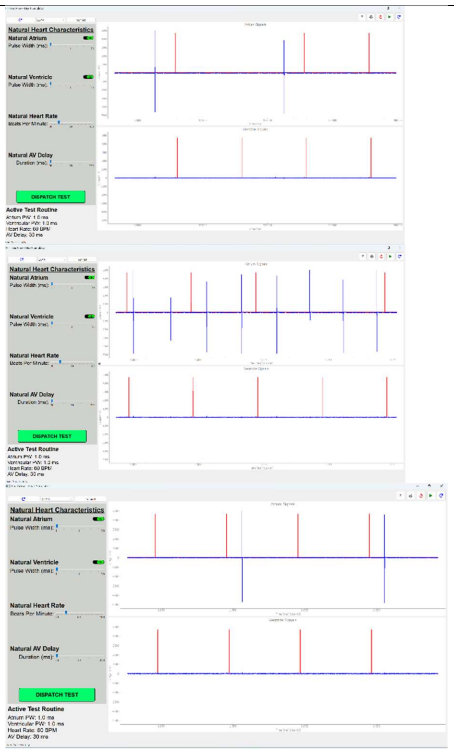
Commented [AN15]: can u include time it takes to speed up and slow down (shoud match recaction time and recovery time)

Mode:	AOOR	
LRL:	80 BPM	
URL:	120 BPM	
MSR:	120 BPM	
Atrial Pulse Amplitude:	5 V	
Atrial Pulse Width:	0.4 ms	
Activity Threshold:	Med	
Response Factor:	8	
Reaction Time:	20 sec	
Recovery Time:	3 min	

		 The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since LRL in this scenario is set rather high the parameters, during shaking the pacemaker will often beat multiple time in between the natural atrium activity as shown in the second screenshot. Due the physical activity the changing_plusetime in the mainstateflow decreases creating a faster BPM as shown in the second picture.

Commented [AN16]: did not mention effect of shaking

Mode:	AOOR	
LRL:	25 BPM	
URL:	180 BPM	
MSR:	180 BPM	
Atrial Pulse Amplitude:	8 V	
Atrial Pulse Width:	2 ms	

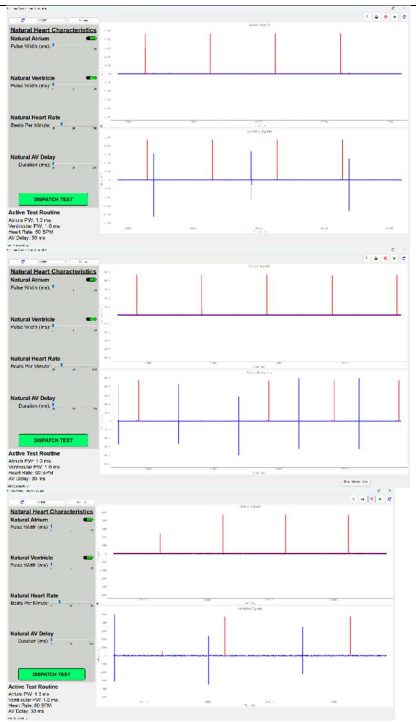
Activity Threshold:	Med	
Response Factor:	8	
Reaction Time:	8 sec	
Recovery Time:	1 min	The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since the LRL in this case is set to 25 the value is clipped to 30 BPM (showcasing our safety features) before the main state flow, this is the same for amplitude (set to 5V), pulse width (set to 1.9ms), URL (set to 175), and MSR (set to 175). So, the first screenshot represents the pacemaker pulsing at 30 bpm. The following one afterwards represents the upper limit it of 175 bpm due to rigours physical activity.

Commented [AN17]: a lot of other params were clicked

VOOR

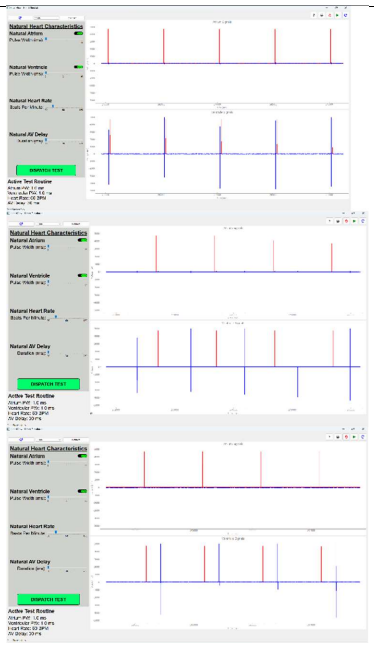
Mode:	VOOR	
LRL:	40 BPM	
URL:	65 BPM	
MSR:	65 BPM	
Ventricular Pulse Amplitude:	3 V	
Ventricular Pulse Width:	1.0 ms	

Activity Threshold:	Low
Response Factor:	4
Reaction Time:	15 sec
Recovery Time:	2 min



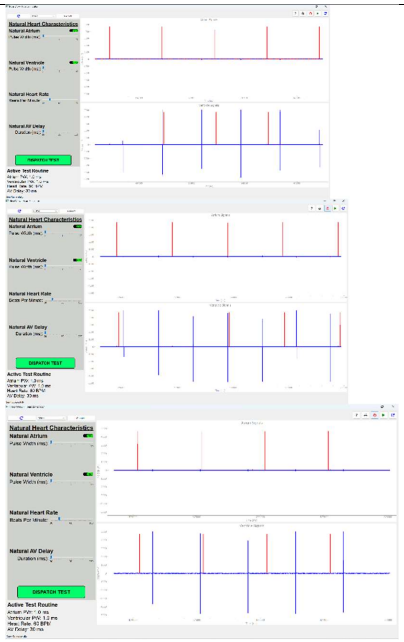
The above screenshots of heart view represent the pre, during, and post physical activity response respectively. As shown in the screenshots the natural BPM is set to 60. During the shake the pulses sent to the atrium only pace ahead of the natural activity by a significant margin. The screenshot proceeding showcases a return to natural atrial pulsing after the 2-minute recovery time.

Mode:	VOOR
LRL:	60 BPM
URL:	80 BPM
MSR:	80 BPM
Ventricular Pulse Amplitude:	4.2 V
Ventricular Pulse Width:	0.9 ms
Activity Threshold:	Med-High
Response Factor:	11
Reaction Time:	10 sec
Recovery Time:	2 min



The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since the LRL is 60 BPM the beats provided are nearly in line with the natural ventricle signals. However, when significant activity is applied to the pacemaker the BPM is affectively raised to 80 with the desired amplitude of 4.2V. Finally, the third shows the recovery from this which is closer to 60 BPM.

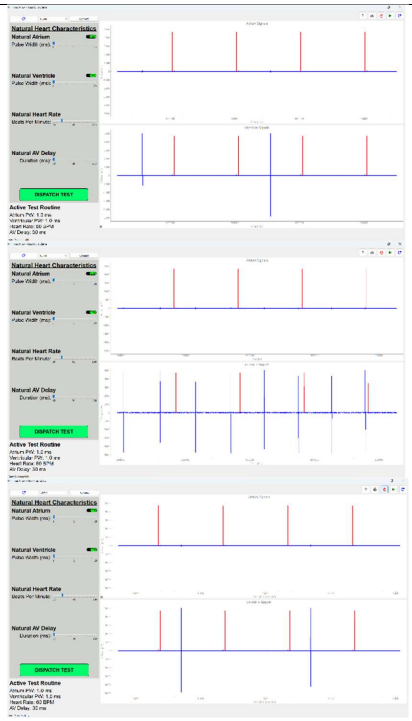
Mode:	VOOR
LRL:	80 BPM
URL:	120 BPM
MSR:	120 BPM
Ventricular Pulse Amplitude:	5 V
Ventricular Pulse Width:	0.4 ms
Activity Threshold:	Med
Response Factor:	8
Reaction Time:	20 sec
Recovery Time:	3 min



The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since LRL, and URL in this scenario is set rather high the parameters, before and during shaking the pacemaker will often beat multiple time in between the natural ventricle activity as shown in the second screenshot. Since recovery is longer the times in the screenshots between the second and third reflect that.

Mode:	VOOR
LRL:	25 BPM
URL:	180 BPM
MSR:	180 BPM

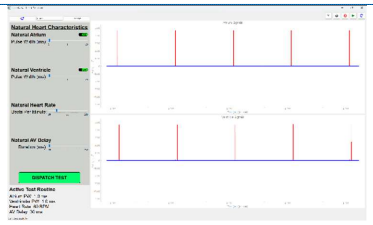
Ventricular Pulse Amplitude:	8 V
Ventricular Pulse Width:	2 ms
Activity Threshold:	Med
Response Factor:	8
Reaction Time:	8 sec
Recovery Time:	1 min



The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since the LRL in this case is set to 25 the value is clipped to 30 BPM (showcasing our safety features) before the main state flow, this is the same for amplitude (set to 5V), pulse width (set to 1.9ms), URL (set to 175), and MSR (set to 175). So, the first screenshot represents the pacemaker pulsing at 30 bpm. The following one afterwards represents the upper limit it of 175 bpm due to rigours physical activity.

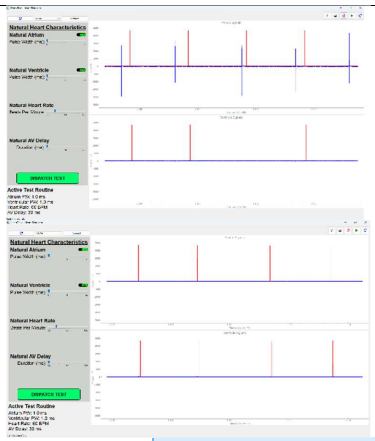
AAIR

Mode:	AAIR
LRL:	40 BPM
URL:	65 BPM
MSR:	65 BPM
Atrial Pulse Amplitude:	3 V
Atrial Pulse Width:	1.0 ms
Atrial Refractory Period:	200 ms



Commented [AN18]: for AAIR and VVIR can we throw in that you can see we dont pulse right after natural (coz we sensing)

Activity Threshold:	Low
Response Factor:	4
Reaction Time:	15 sec
Recovery Time:	2 min

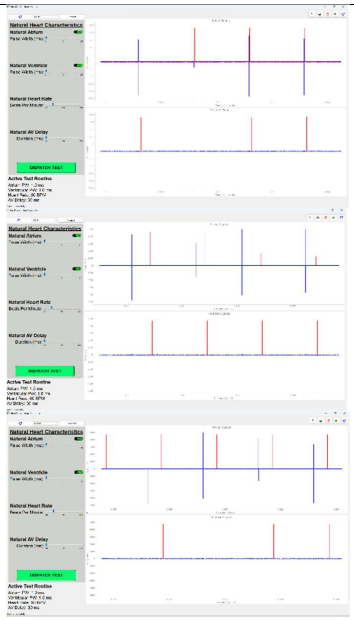


The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since the BPM at resting is sufficient due to the fact that the natural is set to 60 as shown, all pacemaker beats are inhibited.

In other words, the desired bpm is 40 and therefore is lower than the 60 bpm the heart is naturally pumping at so no pulses are asserted. It is only until significant activity is applied which is determined by the activity threshold, to which deliberate pulses can be seen as in the second screenshot. After every pulse a refractory period is accounted for, one of 200ms in this case. After recovery time the activity in the atrium can viewed as it was before shaking.

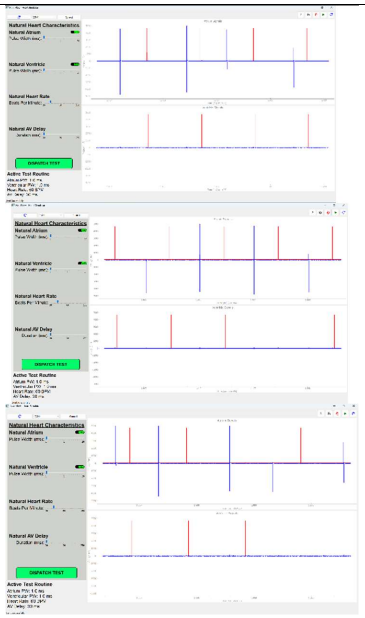
Commented [AN19]: mention tats its set to 40 which is below 60 bpm

Mode:	AAIR
LRL:	60 BPM
URL:	80 BPM
MSR:	80 BPM
Atrial Pulse Amplitude:	4.2 V
Atrial Pulse Width:	0.9 ms
Atrial Refractory Period:	300 ms
Activity Threshold:	Med-High
Response Factor:	11
Reaction Time:	10 sec
Recovery Time:	2 min



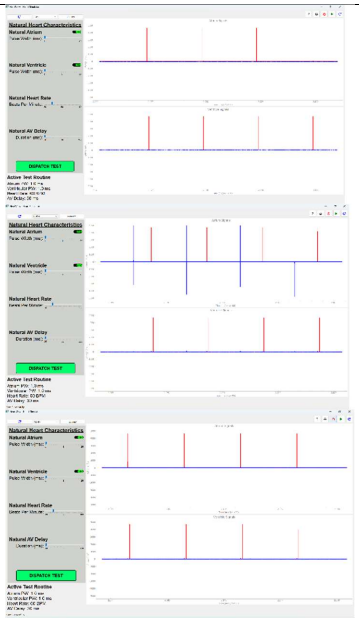
The above screenshots of heart view represent the pre, during, and post physical activity response respectively. It is only when physical activity is applied, above the threshold of “Med-High” that clear pluses in-between natural beats can be viewed. After every pulse a refractory period is accounted for, one of 300ms in this case. Otherwise, atrium activity is viewed as the natural 60 BPM as shown.

Mode:	AAIR
LRL:	80 BPM
URL:	120 BPM
MSR:	120 BPM
Atrial Pulse Amplitude:	5 V
Atrial Pulse Width:	0.4 ms
Atrial Refractory Period:	350 ms
Activity Threshold:	Med
Response Factor:	8
Reaction Time:	20 sec
Recovery Time:	3 min



Since the LRL or desired rate is higher than the preset natural rate the pacemaker pulses at desired intervals to the atrium. After significant physical shaking the pacemaker repeatedly pulses in-between natural beats as shown in the third screenshot. After every pulse a refractory period is accounted for, one of 350ms in this scenario.

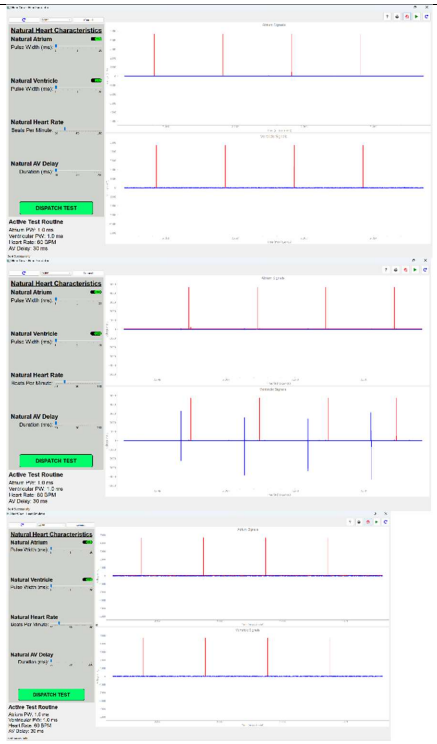
Mode:	AAIR
LRL:	25 BPM
URL:	180 BPM
MSR:	180 BPM
Atrial Pulse Amplitude:	8 V
Atrial Pulse Width:	2 ms
Atrial Refractory Period:	100 ms
Activity Threshold:	Med
Response Factor:	8
Reaction Time:	8 sec
Recovery Time:	1 min



The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since the amplitude in this case is set to 8V the value is clipped to 5V (showcasing our safety features) before the main state flow, this is the same for LRL (set to 30bpm), pulsewidth (set to 1.9ms), URL (set to 175), and MSR (set to 175). So, the second screenshot represents the pacemaker pulsing with an amplitude of 5V, with a refractory period of 100ms, for sensing (inbiting pulse for 100ms after natural pluse), 175bpm, and pluse with of 1.9ms. The following one afterwards represents the upper limit it of 175 bpm due to rigours physical activity.

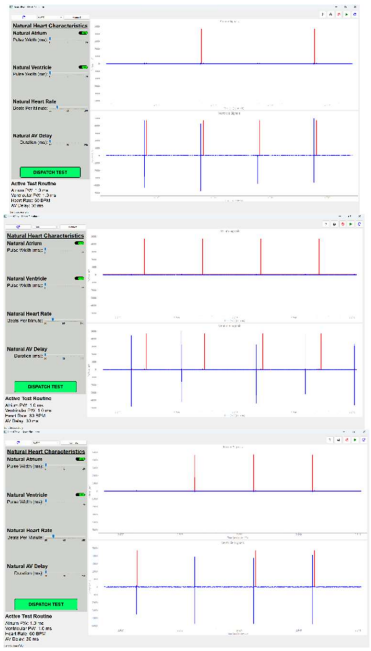
VVIR

Mode:	VVIR
LRL:	40 BPM
URL:	65 BPM
MSR:	65 BPM
Ventricular Pulse Amplitude:	3 V
Ventricular Pulse Width:	1.0 ms
Ventricular Refractory Period:	200 ms
Activity Threshold:	Low
Response Factor:	4
Reaction Time:	15 sec
Recovery Time:	2 min



The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since the BPM at resting is sufficient (the LRL is 40) the natural is set to 60 (greater) as shown, all pacemaker beats are inhibited. It is only until significant activity is applied which is determined by the activity threshold, to which deliberate pulses (of 65 BPM) can be seen as in the second screenshot, each pulse given 200ms after natural pluses. After recovery time the activity in the ventricle can viewed as it was before shaking.

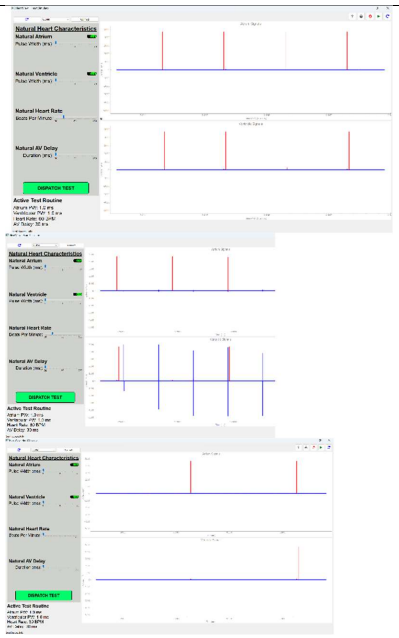
Mode:	VVIR
LRL:	60 BPM
URL:	80 BPM
MSR:	80 BPM
Ventricular Pulse Amplitude:	4.2 V
Ventricular Pulse Width:	0.9 ms
Ventricular Refractory Period:	300 ms
Activity Threshold:	Med-High
Response Factor:	11
Reaction Time:	10 sec
Recovery Time:	2 min



The above screenshots of heart view represent the pre, during, and post physical activity response respectively. It is only when physical activity is applied (which doesn't have to be sustained for a long time due to a 10 second reaction time), above the threshold of "Med-High" that clear pluses in-between natural beats can be viewed. Once again, each pulse is inhibited by 300ms after each pulse as shown in the second screenshot. Otherwise, ventricle activity is viewed as the natural 60 BPM as shown.

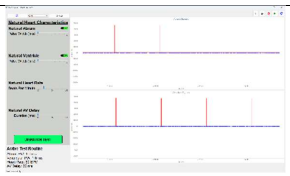
Mode:	VVIR	

LRL:	80 BPM
URL:	120 BPM
MSR:	120 BPM
Ventricular Pulse Amplitude:	5 V
Ventricular Pulse Width:	0.4 ms
Ventricular Refractory Period:	350 ms
Activity Threshold:	Med
Response Factor:	8
Reaction Time:	20 sec
Recovery Time:	3 min



After significant physical shaking the pacemaker repeatedly pulses in-between natural beats as shown in the second screenshot. This combined bpm reaches 120, which pulses with an amplitude of 5V as shown, as other cases have shown the ventricle pulses are inhibited until 350ms after ventricle activity as shown in the second screenshot. Afterwards the ventricle resumes regular functions.

Mode:	VVIR
LRL:	25 BPM
URL:	180 BPM
MSR:	180 BPM
Ventricular Pulse Amplitude:	8 V
Ventricular Pulse Width:	2 ms
Ventricular Refractory Period:	100 ms
Activity Threshold:	Med
Response Factor:	8
Reaction Time:	8 sec

Recovery Time:	1 min	 The above screenshots of heart view represent the pre, during, and post physical activity response respectively. Since the amplitude in this case is set to 8V the value is clipped to 5V (showcasing our safety features) before the main state flow, this is the same for LRL (set to 30bpm), pulse width (set to 1.9ms), URL (set to 175), and MSR (set to 175). So, the second screenshot represents the pacemaker pulsing with the "clipped parameters". The screenshot represents the upper limit it of 175 bpm due to rigours physical activity.

Serial Testing

Test 1 – Pacemaker Serial Transmission Validation

Because integrating directly into the DCM was a large undertaking, the serial components of the pacemaker were first tested with isolated python files. First was a test that data could be received from the pacemaker. This was done by setting known values. The python file was to read and print each received byte.

In this case the following values were set:

Mode	1
LRL	60
URL	120
ATR_PulseAmplitude	3.5
ATR_PulseWidth	0.4
VENT_PulseAmplitude	3.5
VENT_PulseWidth	0.4
VRP	320
ARP	250
MaxSensorRate	120
ReactionTime	30
ResponseFactor	8
RecoveryTime	5
ActivityThreshold	3

Table 6: Set values for serial test.

The following output was received from the pacemaker:

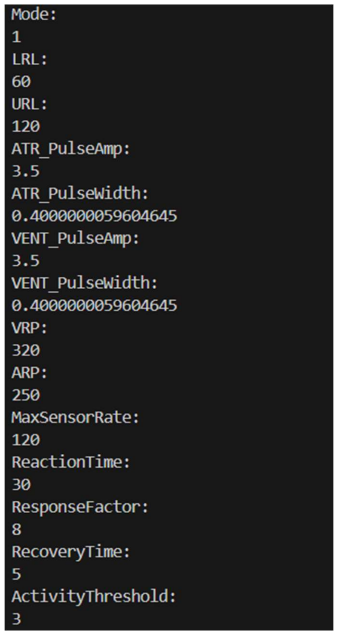


Figure 4524: Terminal output from first serial test.

It is clear that this output matches what was expected apart from some floating point error.

Status: Success.

Test 2 – Serial Receive and Echo Validation

The next key element to be tested is that the pacemaker would receive updated parameters via UART from the python script, update its behaviour accordingly, and echo the parameters back to be printed in the terminal.

The original parameters should be the same as in the above test, and update when a counter is equal to 100.

The parameters should be updated to the following:

Mode	6
LRL	70
URL	100
ATR_PulseAmplitude	2.0
ATR_PulseWidth	1.7
VENT_PulseAmplitude	2.5
VENT_PulseWidth	0.6
VRP	200
ARP	200

MaxSensorRate	130
ReactionTime	15
ResponseFactor	3
RecoveryTime	2
ActivityThreshold	6

Figure 4625: Updated parameters for test 2.

The parameters in the test before sending new parameters to the pacemaker are shown below.

```
iteration # 84
Mode:
1
LRL:
60
URL:
120
ATR_PulseAmp:
3.5
ATR_PulseWidth:
0.4000000059604645
VENT_PulseAmp:
3.5
VENT_PulseWidth:
0.4000000059604645
VRP:
320
ARP:
250
MaxSensorRate:
120
ReactionTime:
30
ResponseFactor:
8
RecoveryTime:
5
ActivityThreshold:
3
```

Figure 4726: Terminal output when counter is equal to 84.

Additionally, heartview shows operation in mode 1, AOO.

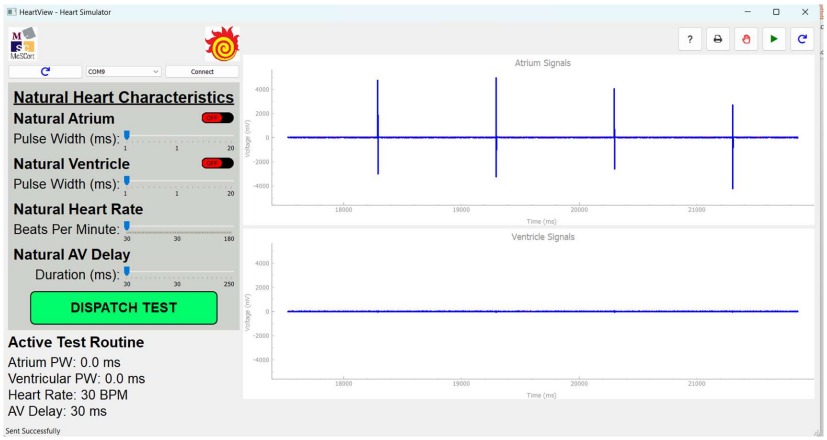


Figure 4827: AOO operation prior to parameter change.

The parameters are shown printed in terminal after updating via a write from UART.

```
iteration # 101
Mode:
6
LRL:
70
URL:
100
ATR_PulseAmp:
2.0
ATR_PulseWidth:
1.7000000476837158
VENT_PulseAmp:
2.5
VENT_PulseWidth:
0.6000000238418579
VRP:
200
ARP:
200
MaxSensorRate:
130
ReactionTime:
15
ResponseFactor:
3
RecoveryTime:
2
ActivityThreshold:
6
```

Figure 4928: Terminal output when counter is equal to 101.

The parameters shown above match the parameters intended to be written to the pacemaker, showing that the correct parameters are received and echoed. The switch to VOOR (mode 6) can also be observed in the heartview image below.

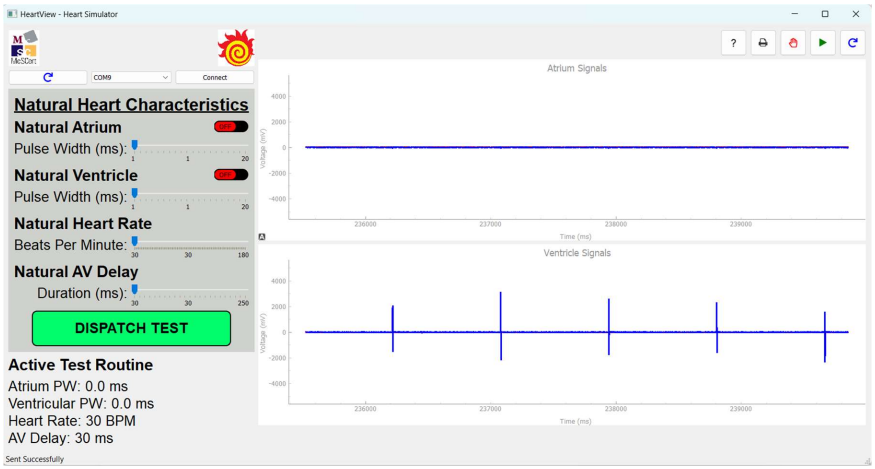


Figure 5029: VOOR operation after parameter change.

Status: Success.

Test 3 – DCM Receive + Graphing

Following the previous 2 tests, proof of concept has been achieved for the entirety of the pacemaker serial system. However, the integration of serial with the DCM still requires testing and the ECG graphing data is near impossible to validate without seeing it graphed.

The ECG should depict what signals are detected on the atrium and ventricle, mirroring the behaviour of heatview. A screenshot of heartview during this test is shown below.

Figure 5130: Heartview during this test.

Notably, there should be approximately 1 pulse per second as shown in heartview. The graphing on the DCM is shown below.

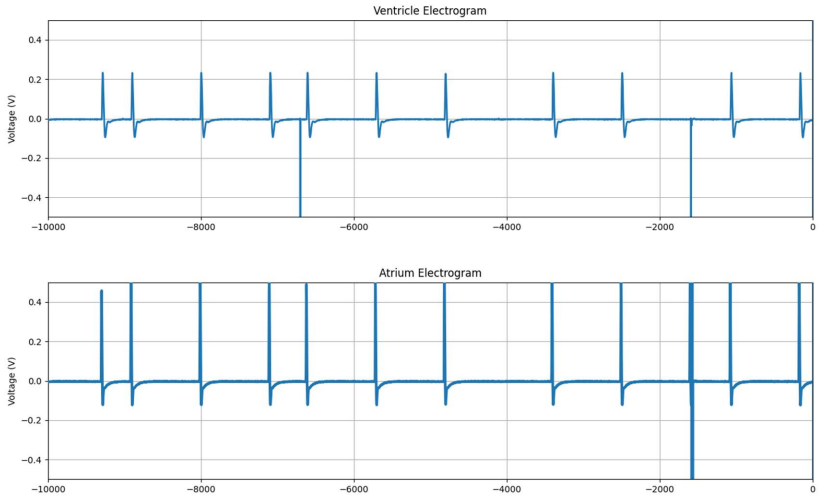


Figure 5231: ECG graphing in DCM.

The proper graphing of both ventricle and atrium data shows that the pacemaker sends proper atrium and ventricle sensing data, and that the DCM is able to properly receive data from the pacemaker.

Status: Success.

Test 4 – Integrated DCM Testing

With the capability of the DCM to read parameters from the DCM established, the last key functionality that needs to be tested is writing over UART from the DCM to the pacemaker. This can be done by setting different modes of operation in the DCM and observing the correct behaviour on heartview. Each test will start from the starting conditions of test 1, which are included below.

Mode	1
LRL	60
URL	120
ATR_PulseAmplitude	3.5
ATR_PulseWidth	0.4
VENT_PulseAmplitude	3.5
VENT_PulseWidth	0.4
VRP	320
ARP	250
MaxSensorRate	120
ReactionTime	30

ResponseFactor	8
RecoveryTime	5
ActivityThreshold	3

Table 7: Original values for DCM testing.

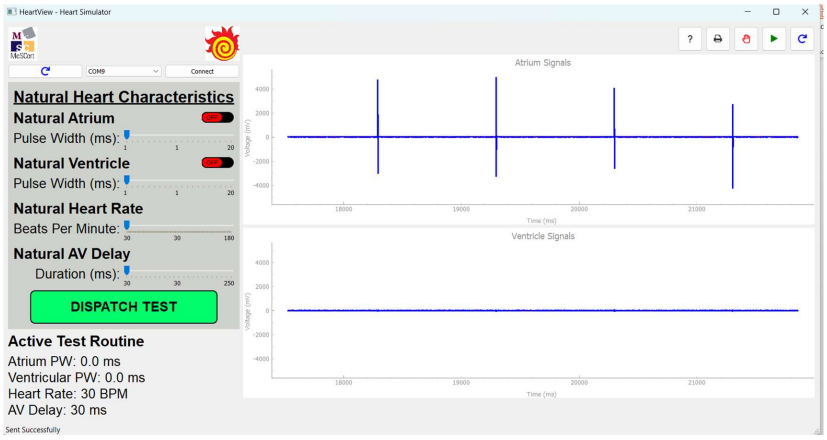


Figure 5332: Original behaviour prior to parameter change.

The following parameters will be used in the first test:

Mode	1 (AOO)
LRL	90
URL	120
ATR_PulseAmplitude	1.5
ATR_PulseWidth	0.1

Table 8: Updated parameters for first integrated test.

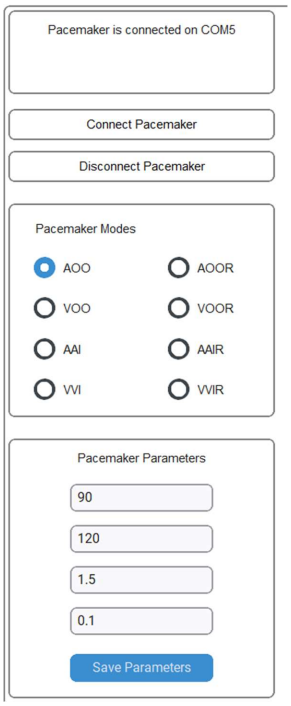


Figure 5433: The parameters are input into the DCM.

The resulting heartview image is shown below.

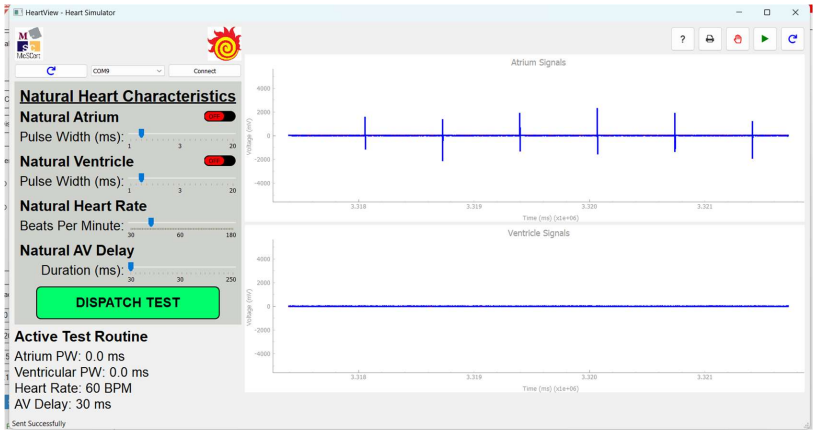


Figure 5534: Updated heartview after first integrated test.

It is clearly visible from the heartview image that the rate has increased to ~90 BPM, and that the amplitude of the pulses has decreased to approximately 1.5V.

The second test will set the following parameters:

Mode	8 (VVIR)
LRL	30
URL	120
VENT_PulseAmplitude	3
VENT_PulseWidth	0.3
VRP	150
MaxSensorRate	140
ReactionTime	20
ResponseFactor	4
RecoveryTime	2
ActivityThreshold	1 (Low)

Table 9: Updated parameters for test 2.

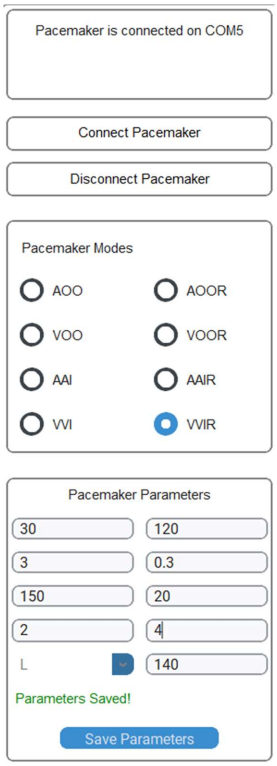


Figure 5635: Parameters set in the DCM.

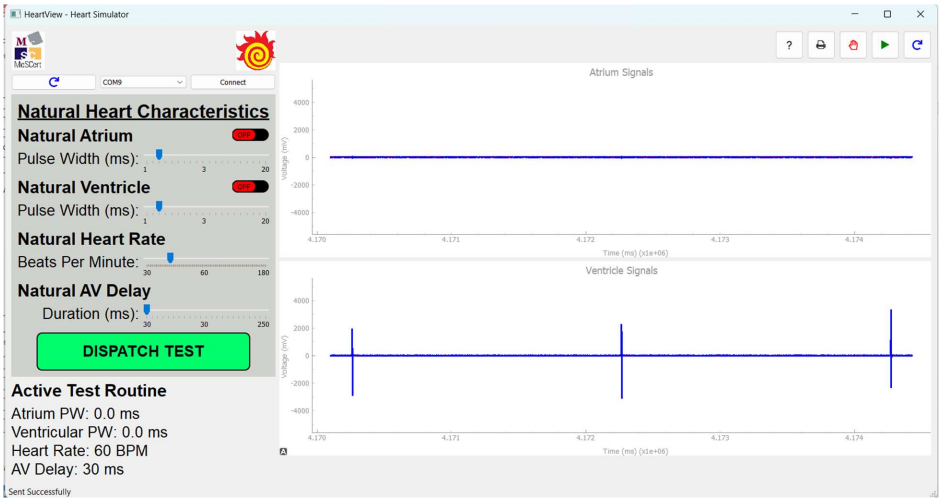


Figure 5736: Heartview after changing parameters from DCM.

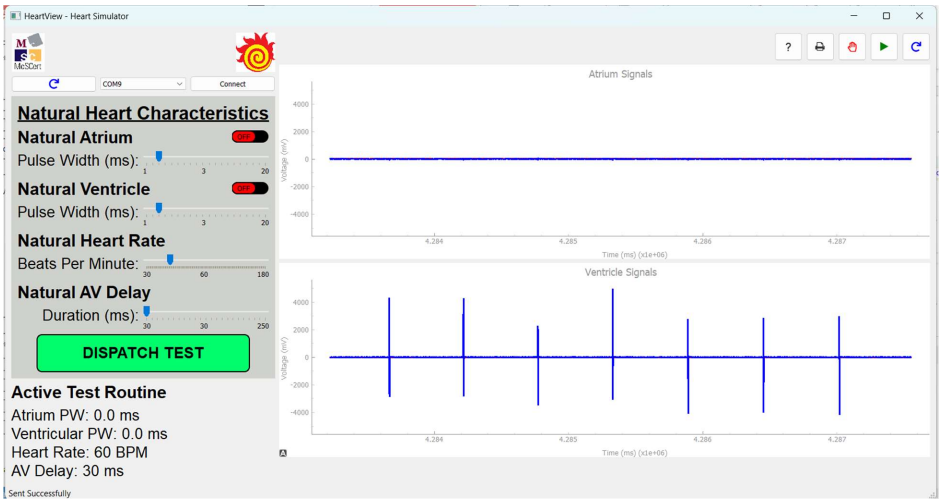


Figure 5837: Heartview after changing parameters, shaking device for ~20s.

The heartview images following parameter changes verifies that the serial transmission from DCM is working properly as the lower rate limit, upper rate limit, pulse amplitude and response time are all easily verifiable. This demonstrates the full working ability of the DCM with serial communication.

Status: Success.

Summary of Testing

ID	Test Case	Expected Result	Observed Result	Pass/Fail
001	AOO, 60 BPM LRL, 0.4ms pulse width, 3V amplitude	Heartview reflects the set test case	As expected	Pass
002	AOO, 80 BPM LRL, 0.6ms pulse width, 4V amplitude	Heartview reflects the set test case	As expected	Pass
003	AOO, 60 BPM LRL, 0.4ms pulse width, 3V amplitude, natural atrium pulses	Heartview reflects the set test case without impact from the natural pulses	As expected	Pass
004	VOO, 60 BPM LRL, 0.4ms pulse width, 3V amplitude	Heartview reflects the set test case	As expected	Pass
005	VOO, 80 BPM LRL, 0.6ms pulse width, 4V amplitude	Heartview reflects the set test case	As expected	Pass
006	VOO, 60 BPM LRL, 0.4ms pulse width, 3V amplitude, natural atrium pulses	Heartview reflects the set test case without impact from the natural pulses	As expected	Pass
007	AAI, 40BPM, natural heart rate set to 60BPM	Only natural beats observed in Heartview	As expected	Pass
008	AAI, 60BPM, natural heart rate set to 60BPM	Only natural beats observed in Heartview	As expected	Pass
009	AAI, 80BPM, natural heart rate set to 60BPM	Pacemaker pulse observed after each natural pulse with a time difference indicative of 80 BPM	As expected	Pass
010	AAI, 65BPM, 100ms refractory period. natural heart rate set to 60BPM	Pacemaker pulse observed after each pulse with time difference indicative of 65 BPM. Refractory period resulting in some pacemaker paces occurring at a 65 BPM interval regardless of natural paces.	As expected	Pass
011	VVI, 40BPM, natural heart rate set to 60BPM	Only natural beats observed in Heartview	As expected	Pass
012	VVI, 60BPM, natural heart rate set to 60BPM	Only natural beats observed in Heartview	As expected	Pass
013	VVI, 80BPM, natural heart rate set to 60BPM	Pacemaker pulse observed after each natural pulse with a time difference indicative of 80 BPM	As expected	Pass
014	VVI, 65BPM, 100ms refractory period, natural heart rate set to 60BPM	Pacemaker pulse observed after each pulse with time difference indicative of 65	As expected	Pass

		BPM. Refractory period resulting in some pacemaker paces occurring at a 65 BPM interval regardless of natural paces.		
015	AOOR, 40 BPM LRL, 60 BPM URL, 3V amplitude, 1 ms width, low activity thresh, response factor 4, 15 sec reaction, natural heart rate set to 60BPM	Heart beats at 40BPM during the pre activity phase, 60BPM during activity (shaking) before recovering to beat at 40BPM once again.	As expected	Pass
016	AOOR, 60BPM LRL, 80 BPM URL, 4.2V amplitude, 0.9 ms width, med-high activity thresh, response factor 11, 10 sec reaction, 2 min recovery time natural heart rate set to 60BPM	Heart beats at 60BPM during the pre activity phase, 80BPM during activity (shaking) before recovering to beat at 60BPM once again.	As expected	Pass
017	AOOR, 80BPM LRL, 120BPM URL, 5V amplitude, 0.4 ms width, med activity thresh, response factor 8, 20 sec reaction, 3 min recovery time natural heart rate set to 60BPM	Heart beats at 80BPM during the pre activity phase, 120BPM during activity (shaking) before recovering to beat at 80BPM once again.	As expected	Pass
018	AOOR, 25BPM LRL, 180BPM URL, 5V amplitude, 0.4 ms width, med activity thresh, response factor 8, 8 sec reaction, 1 min recovery time natural heart rate set to 60BPM	Heart beats at 30BPM during the pre activity phase, 175BPM during activity (shaking) before recovering to beat at 30BPM once again.	As expected	Pass
019	VOOR, 40BPM LRL, 65BPM URL, 3V amplitude, 1 ms width, low activity thresh, response factor 4, 15 sec reaction, 2 min recovery time natural heart rate set to 60BPM	Heart beats at 40BPM during the pre activity phase, 65BPM during activity (shaking) before recovering to beat at 40BPM once again.	As expected	Pass
020	VOOR, 60BPM LRL, 80BPM URL, 4.2V amplitude, 0.9 ms width,	Heart beats at 60BPM during the pre activity phase, 80BPM during	As expected	Pass

	med-high activity thresh, response factor 11, 10 sec reaction, 2 min recovery time natural heart rate set to 60BPM	activity (shaking) before recovering to beat at 60BPM once again.		
021	VOOR, 80BPM LRL, 120BPM URL, 5V amplitude, 0.4 ms width, med activity thresh, response factor 8, 20 sec reaction, 3 min recovery time natural heart rate set to 60BPM	Heart beats at 80BPM during the pre activity phase, 120BPM during activity (shaking) before recovering to beat at 80BPM once again.	As expected	Pass
022	VOOR, 25BPM LRL, 180BPM URL, 8V amplitude, 2 ms width, med activity thresh, response factor 8, 8 sec reaction, 1 min recovery time natural heart rate set to 60BPM	Heart beats at 30BPM during the pre activity phase, 180BPM during activity (shaking) before recovering to beat at 60BPM once again.	As expected	Pass
023	AAIR, LRL 40BPM, URL MSR 65BPM, VenAmp 3V, VenPW 1ms, Ven. Refrac. 200ms, Activ. Thresh. Low, Resp. Fac. 4, Reac. Time 15s, Recov. Time 2min	The heart beats at 40BPM in the pre activity period, rises to 65BPM during the activity period, and returns to 40BPM again.	As expected	Pass
024	AAIR, LRL 60BPM, URL MSR 80BPM, VenAmp 4.2V, VenPW 0.9ms, Ven. Refrac. 300ms, Activ. Thresh. Med-High, Resp. Fac. 11, Reaction Time 10s, Recov. Time 2min	The heart beats at 60BPM in the pre activity period, rises to 80BPM during the activity period, and returns to 60BPM again.	As expected	Pass
026	AAIR, LRL 80BPM, URL MSR 120BPM, VenAmp 5V, AmpPW 0.4ms, Ven Refrac. 350ms, Activ. Thresh Med., Resp. Fac. 8 Reac. Time 20s, Recov. Time 3min	The heart beats at 80BPM in the pre activity period, rises to 120BPM during the activity period, and returns to 80BPM again.	As expected	Pass
027	AAIR, LRL 25BPM, URL MSR 180BPM, AtrAmp 8V, VenPW 2ms, Ven Refrac 100ms, Activity Thresh. Med. Response factor 8,	The heart beats at 80BPM in the pre activity period, rises to 120BPM during the activity period, and returns to 80BPM again.	As expected	Pass

	Reaction time 8s, Recovery time 1m			
028	VVIR, LRL 40BPM, URL MSR 65BPM, VenAmp 3V, VenPW 1ms, Ven. Refrac. 200ms, Activ. Thresh. Low, Resp. Fac. 4, Reac. Time 15s, Recov. Time 2min	The heart beats at 40BPM in the pre activity period, rises to 65BPM during the activity period, and returns to 40BPM again.	As expected	Pass
029	VVIR, LRL 60BPM, URL MSR 80BPM, VenAmp 4.2V, VenPW 0.9ms, Ven. Refrac. 300ms, Activ. Thresh. Med-High, Resp. Fac. 11, Reaction Time 10s, Recov. Time 2min	The heart beats at 60BPM in the pre activity period, rises to 80BPM during the activity period, and returns to 60BPM again.	As expected	Pass
030	VVIR, LRL 80BPM, URL MSR 120BPM, VenAmp 5V, AmpPW 0.4ms, Ven Refrac. 350ms, Activ. Thresh Med., Resp. Fac. 8 Reac. Time 20s, Recov. Time 3min	The heart beats at 80BPM in the pre activity period, rises to 120BPM during the activity period, and returns to 80BPM again.	As expected	Pass
031	VVIR, LRL 25BPM, URL MSR 180BPM, AtrAmp 8V, VenPW 2ms, Ven Refrac 100ms, Activity Thresh. Med. Response factor 8, Reaction time 8s, Recovery time 1m	The heart beats at 25BPM during the pre activity phase, 180BPM during the activity phase, and falls to 25BPM again.	As expected	Pass
032	Pacemaker parameters set as described in the test	Parameters read into the terminal properly by the python file	As expected	Pass
033	Pacemaker parameters set as described in the test, UART write parameters set as described in the test	Parameters read into the terminal properly before and after the write of parameters to the pacemaker	As expected	Pass
034	ECG Data in from the pacemaker	ECG data graphed in the DCM	As expected	Pass
035	Different parameters set in the DCM as described in the test	Behaviour transition and proper behaviour observed in heartview based on the set DCM parameters	As expected	Pass

Considerations

Future Requirement Changes – DCM

The next step for the DCM is communicating serially with the Pacemaker. In real-life applications, the DCM allows physicians to change the operating mode or parameters of the device remotely and non-invasively after implantation. Therefore, since the current iteration of the DCM uses Python, one can implement the PySerial library to interface with the Pacemaker shield. The Pacemaker shield uses Simulink to induce signals, so the project's next stage requires the DCM to receive the output. Furthermore, the pacing modes stimulate the heart chambers individually in the current DCM state. Another implementation must induce signals simultaneously (dual mode).

Given the requirements are likely to change depending on the project's scale, the DCM must be modifiable. For example, the system should handle more than ten users, especially in healthcare settings. If there are more than ten individuals afflicted with bradycardia in one region, cardiologists and other practitioners would have difficulty monitoring them all if the user count was limited. Also, with heart disease becoming more common, medical professionals may call for more sophisticated treatments and, therefore extra pacing modes. The programmable data will change and there must be room for new metrics.

Affected Future Design Decisions – DCM

Currently, the DCM provisions UI space for the Pacemaker status connection but does not display any data. Once the DCM can communicate with the Pacemaker seamlessly, the frame must convey more intricate details. For example, if a new Pacemaker approaches the DCM (it has a different serial number), the DCM should prompt the user of the new device detection. Furthermore, the user data file (users.txt) will have the programmable data, such as the upper rate limit, lower rate limit, atrial amplitude, atrial pulse width, ventricular amplitude, ventricular pulse width, VRP, and ARP, extracted from the TK entries, so iterating through the text file and enumerating all lines will become a lengthy process. Instead, separating the files and printing them in different areas will improve the organization of the folders, therefore avoiding confusion for the developers/users. Even though the primary users of the DCM will be medical professionals, patients will still use the device to view (not alter) their electrograms. Therefore, one future design consideration will be accessibility options for the visually impaired and new languages other than English.

Future Requirement Changes – Pacemaker (After Assignment 1)

There will be additional requirements for the pacemaker in the future. Most notably, the addition of more modes and rate adaptive pacing will be required. This means that the pacemaker must implement the ability to determine the required rate based on accelerometer data, as well as implementing additional operational mode.

Another upcoming major change is the need for input parameters to the pacemaker to be sent serially over USB by the DCM. This means that the data from the DCM will need to be able to receive

this serial data, extract parameters from it, and then set those parameters to be used in the core logic of the pacemaker.

Affected Future Design Decisions – Pacemaker (After Assignment 1)

The structure of the design that comes before the *PacingStateflow* chart will need to be adapted to accommodate the serial data being received from the DCM. This will likely consist of an input subsystem being implemented which will perform necessary extraction of data from the serial input data, as well as performing data validation checks (such as ensuring that all parameters are within a valid range). This makes the proposed input subsystem an immensely important piece of the software for maintaining proper operational safety of the pacemaker.

The additional modes would need to be added, meaning the addition of multiple top level states as well as values to the mode enumeration. In order to facilitate the development of some of these new modes, a new hardware input would be required that allows for input from the on-board accelerometer. Two separate input subsystems may be useful: the one mentioned above for DCM inputs as well as a subsystem for hardware inputs.

Future Requirement Changes – Pacemaker (After Assignment 2)

There are a number of changes that could be made to the requirements of the pacemaker in order to improve its design moving into the future. One item is establishing a handshaking or beginning of communication protocol with the DCM. This would allow key information to be communicated, such as the pacemaker serial number. It would also ensure that a trusted device is reading data from the pacemaker, rather than any device connected to the UART terminal.

An additional future requirement is the implementation of the DDDR mode that paces both the atrium and ventricle in a rate adaptive way while maintaining a proper atrium ventricle delay between paces of the chambers of the heart.

The serial transmission from the pacemaker could be made more efficient. Currently, all parameter and atrium/ventricle graphing data is sent to the DCM every 20ms. While this relieves complexity, it may increase power consumption if that were to be a concern. To address this, the bulk of the packet could be removed (the graphing data) when graphs are not requested.

Affected Future Design Decisions – Pacemaker (After Assignment 2)

The first and third future requirements would be changes to the serial input and output stateflow/subsystem. This development would have to occur alongside DCM development in order to adjust what data is being extracted from a given packet. This would likely require a redesign of the packet structure, where different sync bytes correlate to different structures, or another byte performed this same role.

Implementation of DDDR would require intense modification of the existing pacing stateflow. The implementation of another mode would require additional elements in this case, like the AV delay. The requirement of this AV delay parameter would also necessitate a change in the packet structure to accommodate the addition of this parameter and its transfer between the pacemaker and DCM.

Future Requirement Changes – DCM (After Assignment 2)

There are many requirement changes that can be implemented into the DCM to improve system design. Firstly, the implementation of an emergency shutdown button to instantly kill the system would be useful in preventing medical emergencies in the case of an input mistake.

Additionally, allowing users to import saved presets from their respective JSON file could reduce the amount of time required to configure the pacemaker during a medical emergency. Implementing an input box to reload previous parameters may save precious time that could prevent irreversible damage or death.

Furthermore, many requirement changes from the pacemaker also require changes from the DCM. Implementing methods for serial handshake protocol, DDDR mode compatibility, and differences in graphing timing vs parameter timing are all future changes that can improve the functionality, efficiency, and safety of the integrated pacemaker system.

Affected Future Design Decisions – DCM (After Assignment 2)

Firstly, implementing the kill switch to the system would require small alterations to the user interface, as well as provisioning to ensure that instantly stopping communication with the pacemaker will not disrupt the pacemaker output in any way that might exacerbate the medical condition of the patient. This would include ensuring that the final byte array is successfully transmitted, and the transfer is not halted midway through the process.

Secondly, implementing the saved parameter system would require a method to retrieve the data from the file and display it to the user to ensure it is correct before it is confirmed. Once it is confirmed it can be sent to the pacemaker.

Regarding the pacemaker requirement changes, integrating the handshake protocol would require retrieving and saving the pacemaker serial data to a file and encrypting it for safety. Then the saved serial value can be compared to the connected serial value. Implementing DDDR would require the creation of a new parameter frame and mode entry, as well as the creation of additional parameters to the Mode class and related methods. Finally, differences in transmission timings could be altered in the serial communication file based on various preset flag conditions.

Confirmation of Functionality

Assurance Case (DCM)

The designed assurance case adheres to the safety requirements and specifications outlined in the DCM. One safety precaution occurs when any of the received parameters are empty. For example, if the Pacemaker sends an empty lower rate limit, the DCM sets it to 30 bpm to prevent a potentially fatal catastrophe. Although there are several methods to ensure the received data matches that of the .json file, the DCM needs to fill in all gaps, regardless of whether there's already a form of verification. The assurance case exists in the save_parameters (Figure 59) method within the Window class. Once the function sets and checks the parameters, selects the mode, and prepares the byte array, it'll ensure any saved values are not empty and set to their minimum value. All parameters are saved to their corresponding user and their mode.

```
213  class Window(ctk.CTk):
314      def save_parameters(self):
388          byte_27 = struct.pack('<B', 2 if (self.mode.RecoveryTime == '') else int(self.mode.RecoveryTime))
389          byte_28 = struct.pack('<B', 0 if (self.mode.ActivityThreshold == '') else int(self.activity_thresh_converter(self.mode.ActivityThreshold)))
390
391          byte_arr = byte_0 + byte_1 + byte_2 + byte_3 + bytes4_7 + bytes8_11 + bytes12_15 + bytes16_19 + bytes20_21 + bytes22_23 + byte_24 + byte_25
392          send_data(byte_arr)
393
394          mode_data = {
395              "name": self.mode.name,
396              "LRL": "30" if self.mode.LRL == "" else self.mode.LRL,
397              "URL": "60" if self.mode.URL == "" else self.mode.URL,
398              "AtrAMP": "0.5" if self.mode.AtrAMP == "" else self.mode.AtrAMP,
399              "AtrPW": "0.05" if self.mode.AtrPW == "" else self.mode.AtrPW,
400              "VenAMP": "0.5" if self.mode.VenAMP == "" else self.mode.VenAMP,
401              "VenPW": "0.05" if self.mode.VenPW == "" else self.mode.VenPW,
402              "ARP": "150" if self.mode.ARP == "" else self.mode.ARP,
403              "VRP": "150" if self.mode.VRP == "" else self.mode.VRP,
404              "ReactionTime": "10" if self.mode.ReactionTime == "" else self.mode.ReactionTime,
405              "RecoveryTime": "2" if self.mode.RecoveryTime == "" else self.mode.RecoveryTime,
406              "ResponseFactor": "1" if self.mode.ResponseFactor == "" else self.mode.ResponseFactor,
407              "ActivityThreshold": "VL" if self.mode.ActivityThreshold == "" else self.mode.ActivityThreshold,
408              "MaxSensorRate": "50" if self.mode.MaxSensorRate == "" else self.mode.MaxSensorRate
409          }
```

Figure 5938: filling in gaps in parameters