SE 3K04

Pacemaker Project

Assignment 2

Group 11

Divesh Chudasama

Caleb Gannon

Ryan Lee

Ben Stephens

Gordon Zhang

Table of Contents

Introduction

The Pacemaker shield is a device used to artificially simulate a heart in real-time. It will be the bioelectrical interface between the heart and the microcontroller, and the second assignment will build on the pacemaker functionality implemented in Assignment 1. There will be more modes implemented, the DCM will be improved, and electrograms will be introduced.

Overview and Components

There are 3 components to this assignment with an added bonus. Part 1 involves the pacemaker design, we are building on the previous modes implemented in assignment 1 and implementing DOO as well as the rate responsive feature for bonus marks. By setting certain inputs into the FSM, the code will generate output signals to be displayed on the oscilloscope for pacing. Additionally, we will also need a high level design of how the accelerometer readings will affect our rate. Part 2 involves the DCM, which handles the communication to the pacemaker. It will have the interface with a welcome screen and the ability to register users and login as existing users. This is where we must expand it to include the additional modes from this assignment as well as mode switching for communication with the device. Additionally, hardware hiding must be implemented in our design, mostly in the Simulink model. Part 3 is testing, we must have sufficient test cases to prove the correctness of the design. Lastly, the bonus is composed of the accelerometer rate adaptive mode, as well as the presentation and serial transmission of the atrial and ventricular egram data from the Pacemaker to the DCM..

Pacemaker Design

There are a multitude of modes to implement for the Pacemaker on Simulink. AOO, VOO, AAI, and VVI from assignment 1, as well as DOO and the rate adaptive versions of each of the previous modes. Based on the stateflow and parameters given in the requirements document, the FSM must transition between the modes and repeat the cycle. In terms of the parameters, the most important are the pulse characteristics of width and amplitude, rate characteristics of limits and delays, and the specific chambers being paced. The states and its transitions will be reflected in the colour of the on-board LEDs.

Modes and States

AOO

Capacitors are charged and then discharged at a variable BPM. C22 is the onboard capacitor that is used to charge the heart. C21 is used to then collect that charge once the pace has been delivered to the heart.

- Atrial Pacing
- No sensing

Note: All pacing cycle inputs are boolean except PACING_REF_PWM, which accepts an integer from 0-100.

Table 1: AOO Specific Variables

	sensing (front_ctrl)	atr_vent	BPM	Pulse_width (ms)	Amplitude*
Example Input:	false	true	60	30	100

*amplitude is an int which sets the duty cycle of the PWM.

Table 2: AOO Charge Cycle

Pin	Setting
ATR_PACE_CTRL	LOW
VENT_PACE_CTRL	LOW
PACING_REF_PWM	2000 Hz with Desired Duty Cycle
PACE_CHARGE_CTRL	HIGH

Table 3: AOO Discharge Cycle

Pin	Setting
PACE_GND_CTRL	HIGH
VENT_PACE_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
ATR_PACE_CTRL	LOW
ATR_GND_CTRL	HIGH
VENT_GND_CTRL	LOW

Note: Charge and Discharge Cycles run simultaneously.

Table 4: AOO Pace Cycle

Pin	Setting
PACE_CHARGE_CTRL	LOW
PACE_GND_CTRL	HIGH
ATR_PACE_CTRL	HIGH
ATR_GND_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
VENT_GND_CTRL	LOW
VENT_PACE_CTRL	LOW

VOO

Capacitors are charged and then discharged at a variable BPM. C22 is the onboard capacitor that is used to charge the heart. C21 is used to then collect that charge once the pace has been delivered to the heart.

- Ventricular pacing
- No sensing

Note: All pacing cycle inputs are boolean except PACING_REF_PWM, which accepts an integer from 0-100.

Table 5: VOO Specific Variables

	sensing (front_ctrl)	atr_vent	BPM	Pulse_width (ms)	Amplitude*
Example Input:	false	false	60	30	100

*Amplitude is an int which sets the duty cycle of the PWM.

Table 6: VOO Charge Cycle

Pin	Setting
ATR_PACE_CTRL	LOW
VENT_PACE_CTRL	LOW
PACING_REF_PWM	2000 Hz with Desired Duty Cycle
PACE_CHARGE_CTRL	HIGH

Table 7: VOO Discharge Cycle

Pin	Setting
PACE_GND_CTRL	HIGH
VENT_PACE_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
ATR_PACE_CTRL	LOW
ATR_GND_CTRL	LOW
VENT_GND_CTRL	HIGH

Note: Charge and Discharge Cycles Applied Simultaneously

Table 8: VOO Pace Cycle

Pin	Setting
PACE_CHARGE_CTRL	LOW
PACE_GND_CTRL	HIGH
ATR_PACE_CTRL	LOW
ATR_GND_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
VENT_GND_CTRL	LOW
VENT_PACE_CTRL	HIGH

AAI

While sensing, the atr_cmp_detect looks for a pulse and then waits until the lower bmp limit has passed. If nothing is detected then a pace is delivered, otherwise it remains in sensing mode.

- Atrial pacing
- Atrial sensing

Note: All pacing cycle inputs are boolean except PACING_REF_PWM, which accepts an integer from 0-100.

Table 9: AAI Specific Variables

	sensing (front_ctrl)	set_atr_threshold	atr_vent	BPM	Pulse_width (ms)	Amplitude *
Example Input:	true	95	true	60	30	100

*Amplitude is an int which sets the duty cycle of the PWM.

Table 10: AAI Charge Cycle

Pin	Setting
ATR_PACE_CTRL	LOW
VENT_PACE_CTRL	LOW
PACING_REF_PWM	2000 Hz with Desired Duty Cycle
PACE_CHARGE_CTRL	HIGH

Table 11: AAI Discharge Cycle

Pin	Setting
PACE_GND_CTRL	HIGH
VENT_PACE_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
ATR_PACE_CTRL	LOW
ATR_GND_CTRL	HIGH
VENT_GND_CTRL	LOW

Note: Charge and Discharge Cycles run simultaneously.

Table 12: AAI Pace Cycle

Pin	Setting
PACE_CHARGE_CTRL	LOW
PACE_GND_CTRL	HIGH
ATR_PACE_CTRL	HIGH
ATR_GND_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
VENT_GND_CTRL	LOW
VENT_PACE_CTRL	LOW

VVI

While sensing, the atr_cmp_detect looks for a pulse and then waits until the lower bmp limit has passed. If nothing is detected then a pace is delivered, otherwise it remains in sensing mode.

-Ventricular Pacing

-Ventricular Sensing

Table 13: VVI Specific Variables

	sensing (front_ctrl)	set_vent_threshold	atr_vent	BPM	Pulse_width (ms)	Amplitude*
Example Input:	true	95	false	60	30	100

*Amplitude is an int which sets the duty cycle of the PWM.

Note: All pacing cycle inputs are boolean except PACING_REF_PWM, which accepts an integer from 0-100.

Table 14: VVI Charge Cycle

Pin	Setting
ATR_PACE_CTRL	LOW
VENT_PACE_CTRL	LOW
PACING_REF_PWM	2000 Hz with Desired Duty Cycle
PACE_CHARGE_CTRL	HIGH

Table 15: VVI Discharge Cycle

Pin	Setting
PACE_GND_CTRL	HIGH
VENT_PACE_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
ATR_PACE_CTRL	LOW
ATR_GND_CTRL	HIGH
VENT_GND_CTRL	LOW

Note: Charge and Discharge Cycles run simultaneously.

Table 16: VVI Pace Cycle

Pin	Setting
PACE_CHARGE_CTRL	LOW
PACE_GND_CTRL	HIGH
ATR_PACE_CTRL	HIGH
ATR_GND_CTRL	LOW
Z_ATR_CTRL	LOW
Z_VENT_CTRL	LOW
VENT_GND_CTRL	LOW
VENT_PACE_CTRL	LOW

DOO

This mode involves the asynchronous pacing of both the atrium and ventricle. The new parameter DOO is a boolean that is set to true when in DOO mode.

Table 17: DOO Specific Variables

	sensing (front_ctrl)	DOO	BPM	Pulse_width (ms)	Amplitude*
Example Input:	false	true	60	30	100

*Amplitude is an int which sets the duty cycle of the PWM.

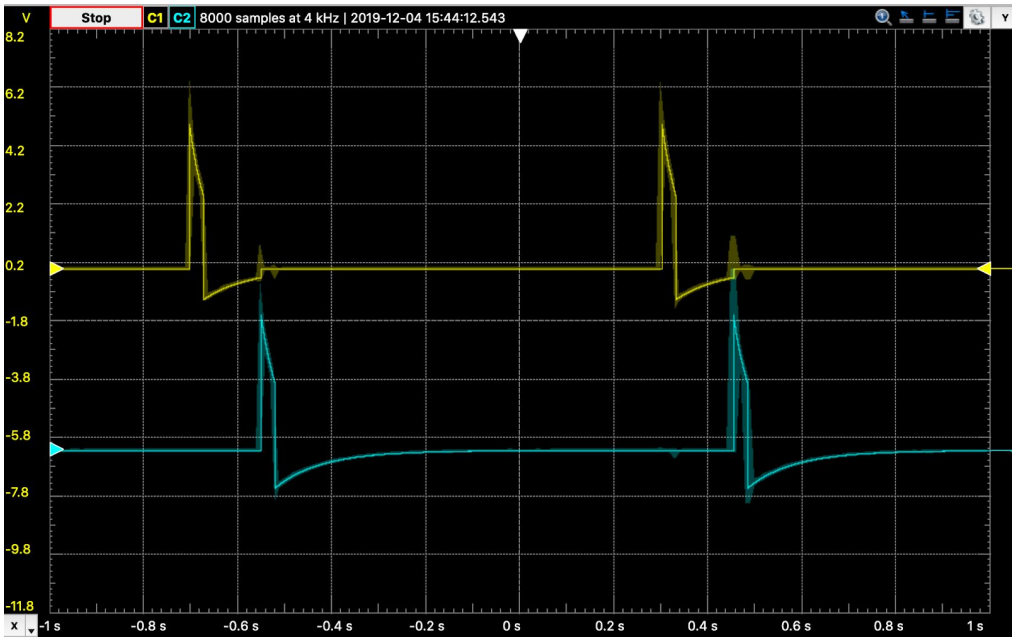


Figure 1: DOO Oscilloscope Graph

As you can see from **Figure 1**, there is a slight delay between the atrium signal (yellow) and ventricle signal (blue). This AV delay comes from the physical limitation of the heart, stemming from the fact that the valves in each chamber have to open and close. The delay can be calculated by taking the time difference between the two waves.

Assignment 1 Stateflow

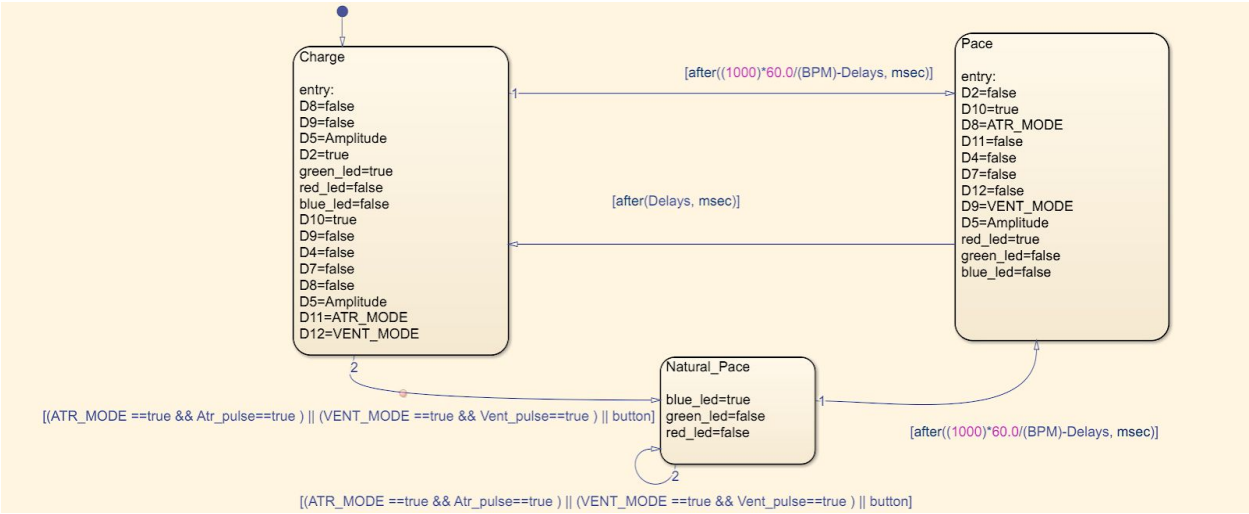


Figure 2: Chart Stateflow for Assignment 1

The three states are Charge, Pace and Natural_Pace. The "charge" state charges C22 while simultaneously discharging C21, and opens the switches responsible for pacing the heart. After a delay we transition into the "Pace" state.

The "Pace" state opens the switches responsible for pacing the heart, while closing pace_charge_ctrl to ensure the heart is never directly connected to the PWM source. It will transition to the "Charge" state after a delay that is defined by the inputs. This delay is equivalent to the **pulse width**.

The "natural_pace" is the state where the system looks for a natural heart pulse, it checks for 3 conditions to be true, if the system is in AAI, it waits for a signal from the atrium. In VVI, it waits for a signal from the ventricle. Also, it checks if the push button was pressed **(bonus)**. If the lower bpm limit is exceeded and no pulse from the heart is detected, the FSM transitions back to the "Pace" state.

The **lower bpm limit** is handled by the input "BPM". The code converts BPM into ms delay, and this delay is adjusted for the pulse width then used in the "after" function. Our code returns to the automated cycle after it has been in the natural_pace state for greater than the lower bpm limit allows.

Assignment 2 Stateflow

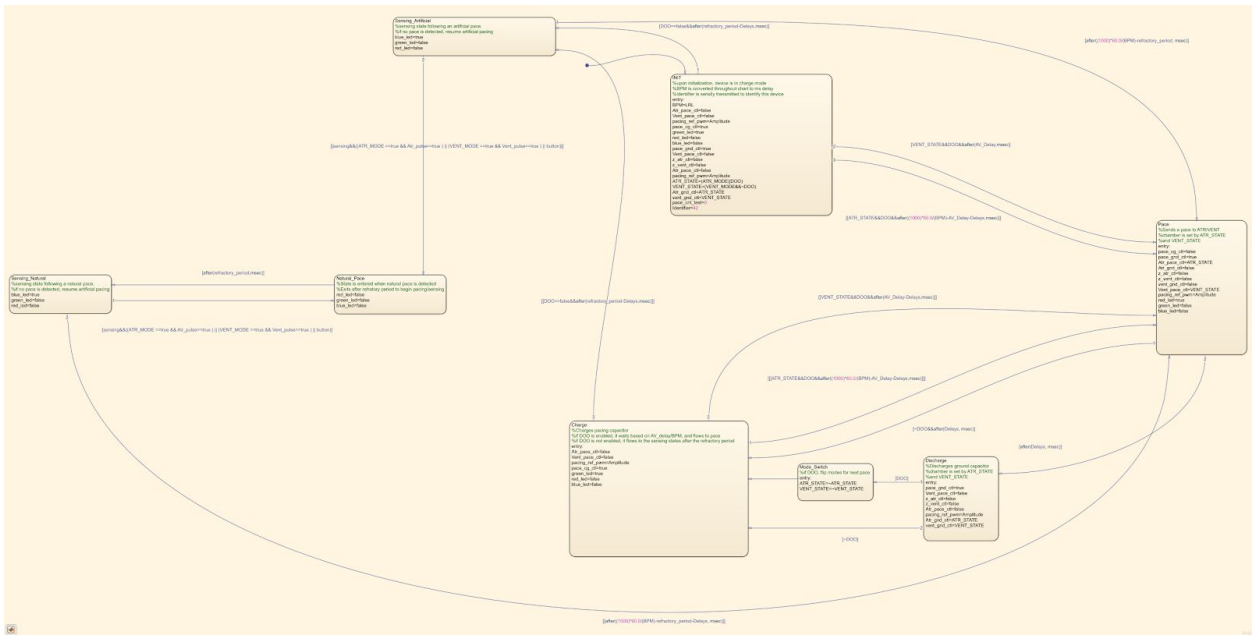


Figure 3: Pacemaker Stateflow for Assignment 2

There are two main flows that exist within the chart. One is enabled with DOO, and the other is for the other four modes. The DOO flow runs between charge, pace, and discharge, switching between atrium and ventricle between discharge and charge. This allows for a seamless DOO flow, with all programmable parameters such as av_delay and BPM only affecting the duration of state changes. If DOO is not enabled, then after charging the pacemaker will wait the refractory period duration, then enter a sensing state. If sensing is disabled for VOO and AOO, the sensing state is a waiting state for the duration of the BPM delay. After the BPM delay a pace is applied, the ground capacitor discharged, and the cycle continues. If sensing is enabled, while the pacemaker is in the sensing state it can detect a sense from its enabled chamber of the heart. If a natural pace is detected, it enters the *natural pace* state. It waits the refractory period in this state, and then resumes sensing. If a sense is detected, it returns to the natural pace state. If nothing is detected, the pacemaker begins to once again automatically pace, and the cycle continues.

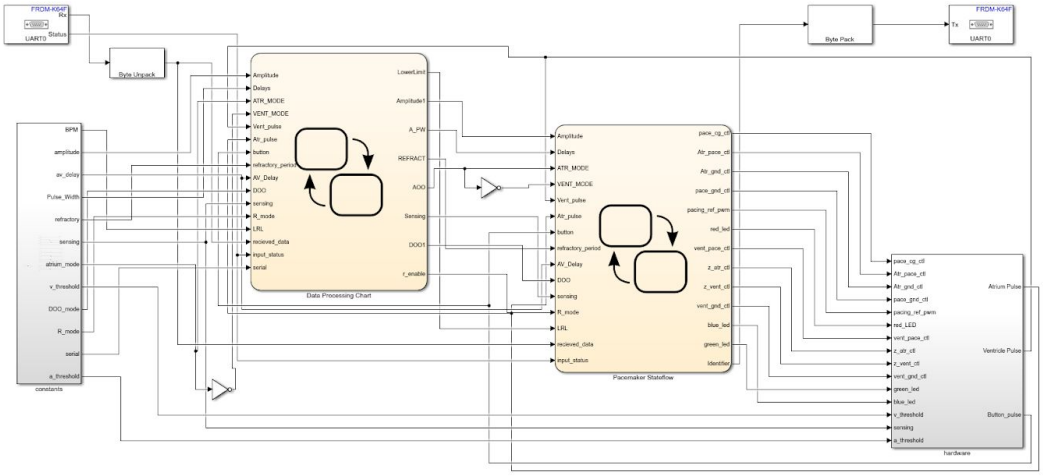


Figure 4: Simulink Design

The overall simulink model ties the serial processing and the predefined constants together in the Data Processing Chart, and transmits the desired parameters to the stateflow. The stateflow effectively runs the pacemaker, by transmitting the desired operations to the microcontroller, through the hardware hiding module. The stateflow also emits an identifier which is serially transmitted to the DCM, to identify that our specific design has been connected.

Hardware Hiding

Table 18: Input Pins Descriptions and Functionality

	#	Pin Name	Pin	Corresponding Name	Simulink Name	Functionality
Inputs	1	D2	PTB9	PACE_CHARGE_CTRL	pace_cg_ctl	Switch: Used to start/stop charging of C22 capacitor.
	2	D8	PTC12	ATR_PACE_CTRL	Atr_pace_ctl	Switch: Used to discharge C22 through atrium.
	3	D11	PTD2	ATR_GND_CTR	Atr_gnd_ctl	Switch: Used to

				L		connect ATR_RING_OUT to GND.
	4	D10	PTD0	PACE_GND_CTRL	pace_gnd_ctl	Switch: Used to allow current to flow from ring to tip.
	5	D5	PTA2	PACING_REF_PWM	pacing_ref_pwm	Used to charge C22.
	6	red_LED	RED_LED			Pacing Mode
	7	D9	PTC4	VENT_PACE_CTRL	vent_pace_ctl	Switch: Used to discharge C22 through ventricle.
	8	D4	PTB23	Z_ATR_CTRL	z_atr_ctl	Used to analyze impedance of atrial electrode.
	9	D7	PTC3	Z_VENT_CTRL	z_vent_ctl	Used to analyze impedance of ventricle electrode.
	10	D12	PTD3	VENT_GND_CTRL	vent_gnd_ctl	Switch: Used to discharge blocking capacitor through ventricle to prevent charge buildup.
	11	green_led	GREEN_LED			Charging Mode

	12	blue_led	BLUE_LED			Natural_Pace Mode
	13	threshold*	PTA1/ PTC2			Sets voltage threshold of sensor. One variable for atrium and ventricle.
	14	sensing	PTD1			
Outputs	1	Atrium Pulse	PTC16			Returns boolean for Pulse detected in atrium
	2	Ventricle Pulse	PTC17			Returns boolean for Pulse detected in ventricle
	3	Button_Pulse				Returns boolean for a detected button press

*Threshold has 2 pins as one is for atrium and one ventricle.

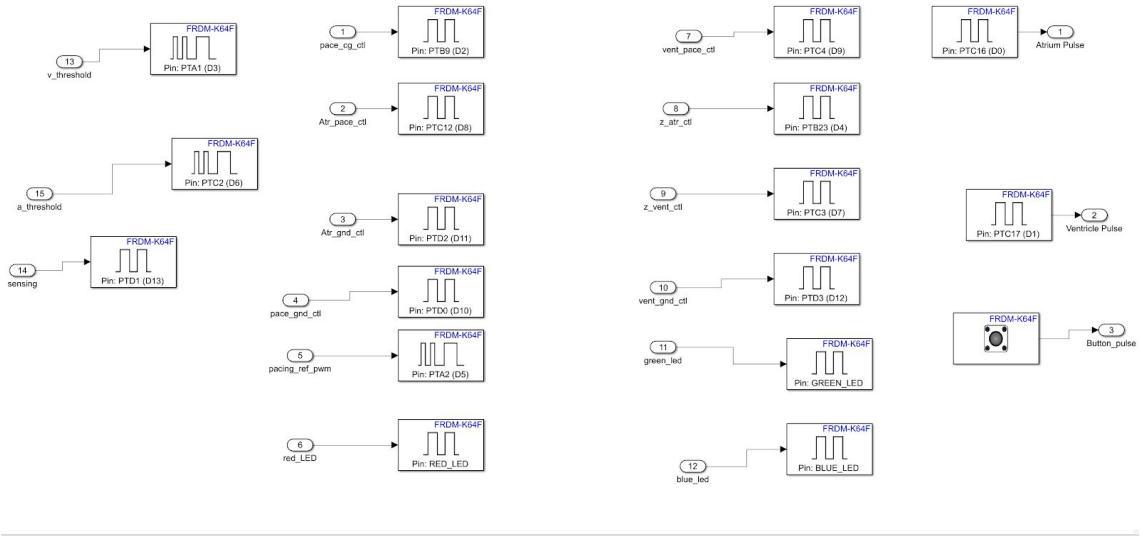
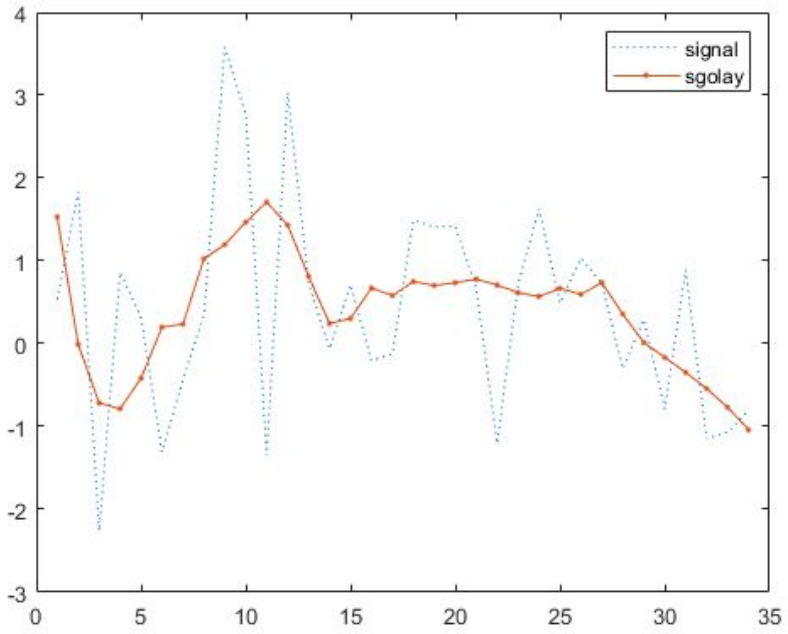


Figure 5: Implementation of Hardware Hiding

Rate Adaptive Mode with Accelerometer

In order to implement the rate adaptive modes for our pacemaker, an accelerometer is needed to detect activity rates of the individual. The problem is, the signal can have a ton of noise which may reduce the accuracy of the rate adaptive modes unless some form of signal processing is done to it to determine the steady-state patient activity. The signal processing algorithm that we chose to employ is known as the Savitzky-Golay Filter. It approximates using convolution, by performing a least squares fit of a small set of consecutive points to a polynomial. This process repeats for the rest of the signal until a set of polynomials is generated to model the entire input signal, which produces the smoothed output signal as a set of data points based on the central point of the polynomials. This can be implemented in our project using the built in matlab function “sgolayfilt()” on the accelerometer signal.



https://www.mathworks.com/help/signal/ref/sgolayfilt.html

Figure 6: Original vs Smoothed Signal Using Savitzky-Golay Filter

To create the rate adaptive mode using the now smoothed data, we would first identify through testing an ideal activity threshold level. If the filtered data crossed this threshold, the device would enter into an adaptive state. The adaptive state would now vary the BPM based on a few parameters. Firstly, we would subdivide activity level values, time durations, and BPM values, up to the upper rate limit. Each BPM increment would correspond to an activity level, and a time value. For example, if a certain activity level is reached, this would correspond to a new BPM. But the longer the patient stays in this activity level, the device will move up to correspondingly higher BPM values. Similarly, if the device is in a higher BPM threshold, but the device activity levels drop, the BPM will drop incrementally depending on the time in the lower activity levels. The effect that time has on the rate of BPM dropping would be proportional to the difference between the current BPM, and the BPM assigned to that activity level, multiplied by a constant. This is known as the response factor. This works the same for rising activity levels. Also note, we would consider any activity below the threshold as one increment, with an assigned BPM equal to the lower rate limit. This would be considered when calculating response time and recovery time, or the duration to go from the lower rate limit to the maximum sensor rate, or vice versa.

In terms of sampling the data, there has to be a certain rate at which we do to prevent aliasing. This comes from the Nyquist frequency, which is at minimum 2x the frequency of the input signal. By sampling at the Nyquist frequency, we ensure that all signals are detected correctly and not ignored.

Simulink Serial Communication Protocol

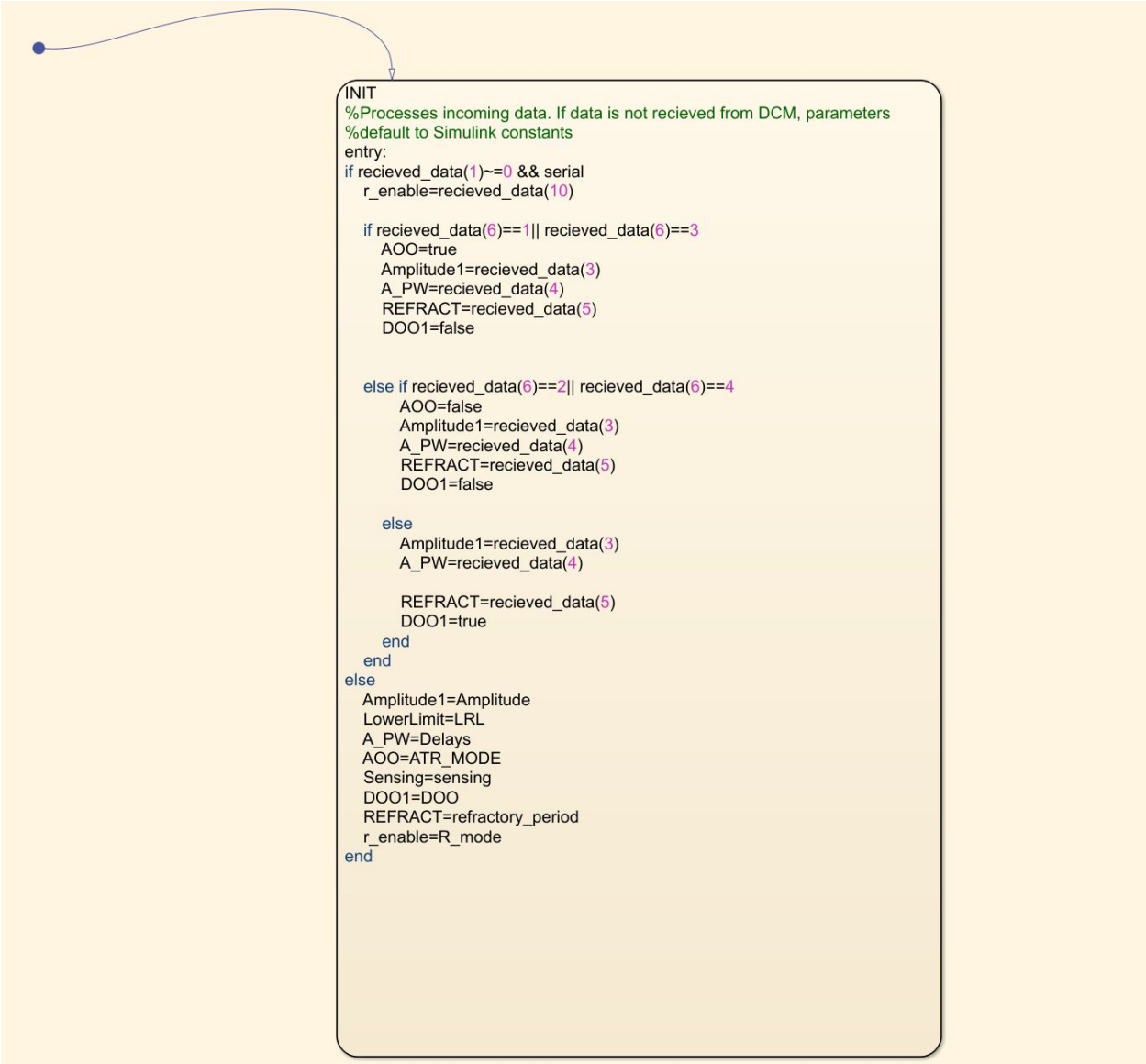


Figure 7: Simulink Serial Chart

The data is processed on the simulink side by first checking for incoming serial data. In the event of no incoming data, the device defaults to its preset constants. In the event of incoming data, the chart assigns

the respective incoming vector indices to their desired output, in accordance with the protocol established by the DCM , found on page 31.

Requirements Changes

Assignment 1

In terms of requirements that may change, right now we have a number of values for input that we arbitrarily picked for the testing of the Pacemaker design. Right now our input requirements are BPM, Amplitude, Limits, Pulse-Width, Escape Interval, Sensing, Atr_vent, and Threshold. These will eventually be replaced by values like Lower Rate Limit, Atrial Amplitude, Atrial Pulse Width, Ventricular Amplitude, Ventricular Pulse Width, VRP, and ARP which will be set from the DCM itself through a custom designed interface. As we get closer to completing the project, certain limitations may also arise as we understand more about the functionality of the Pacemaker. This could result in changes to pacing and input values for our Pacemaker. Finally, during our demo it was discovered that we did not fully implement our refractory period for our natural pace mode. We implemented this and correctly used two different threshold values for our simulink model instead of one.

Assignment 2

In terms of requirements for assignment 2 that may change, currently we only have DOO implemented as well as the assignment 1 modes. We would also implement the rate responsive versions of all the previous modes through the use of an accelerometer to sense changes in physical activity and modify the pacing rate accordingly depending on if the user is walking, jogging, or running. We would do this using a smoothing function to interpret the data, and assigning different activity levels to measured accelerometer data, relative to time. This is further explained above under **Figure 6**. The AV delay is in place due to the physical limitations of the heart. Since there is a delay between natural paces of the atrium and ventricle for the flow of blood. This delay is also required for the DOO mode, where we pace both the atrium and ventricle. In terms of parameters that we have not used but would have implemented, we did not actually use any parameters linked to rate adaptive modes as mentioned before. This would include Activity Threshold, Reaction Time, Response Factor, Recovery Time and Hysteresis.

Design Changes

Assignment 1

In assignment 1, our Simulink design was simple as we initially had three states. However, as we implemented the refractory period we missed and additional conditions, our design became confusing and messy. To fix this we renamed and reordered the states to better reflect their function and in some cases combined similar states to simplify our model. This made designing assignment 2 much easier as that added many more things to the design. If we didn't clean up assignment 1 beforehand then it would've been difficult to design and debug assignment 2. We also renamed our pins from their pin number to their actual functionality. This is important as it makes it easier to see what each pin does which would therefore make analyzing and debugging easier. We also added annotations to our states, to help with code readability.

Assignment 2

Future changes to code in following assignment 2 would be the inclusion of hysteresis pacing for sensing modes. This would be done by assigning a boolean to represent if hysteresis mode is enabled, and in the case that it is a seperate variable would set the escape interval the pacemaker would wait to resume artificial pacing, following a natural pace. We would finalize our serial communication design. During the demo it was noted that our current attempts employed a sacrifice in resolution to transmit each of the DCM data as one byte, by dividing values greater than 255 by 2, and multiplying back within the pacemaker. This was a workaround due to the time constraints of the demo and hardware failures of our device, and in full practice we would implement the transmission of data as floats. This allows for the use of the full range of rational numbers on either side of the pacemaker/DCM interaction. Optimistically speaking, we would hope to implement the full range of modes given in the pacemaker documentation through to DDDR.

Rate Adaptive Mode

To implement rate adaptive mode we would have a state within *pacemaker stateflow* be grabbing acceleration values at a fixed sample rate. These values would be filtered and processed within this state, and the result of their processing would modify the current BPM value within the stateflow. The processing methods are further explained above, below figure 3 on page 16.

Testing

Table 19: Testing of Simulink Model

Purpose	Input	Expected Output	Actual Output	Result (Pass/Fail)
AOO operation	Lower rate limit: 60 Upper rate limit: 60 Amplitude: 100 Pulse width: 30	Pace Atrium at 60 bpm regardless of sensing	Pace Atrium at 60 bpm regardless of sensing	Pass
VOO operation	Lower rate limit: 60 Upper rate limit: 60 Amplitude: 100 Pulse width: 30	Pace Ventricle at 60 bpm, no sensing	Pace Ventricle at 60 bpm, no sensing	Pass
AAI operation	Lower rate limit: 60 Upper rate limit: 60 Amplitude: 100 Pulse width: 30	Sense for natural pace, provide Atrium pace when no natural pace detected	Sense for natural pace, provide Atrium pace when no natural pace detected	Pass
VVI operation	Lower rate limit: 60 Upper rate limit: 60 Amplitude: 100 Pulse width: 30	Sense for natural pace, provide ventricle pace when no natural pace detected	Sense for natural pace, provide ventricle pace when no natural pace detected	Pass

DOO operation	Lower rate limit: 60 Upper rate limit: 60 Amplitude: 100 Pulse width: 30	Pace both atrium and ventricle at 60 bpm	Pace both atrium and ventricle at 60 bpm	Pass
String Input	Lower rate limit: 60 Upper rate limit: 60 Amplitude: “cc” Pulse width: 30	Type error	Matlab error: Expected int16	Pass
Negative Input	Lower rate limit: 60 Upper rate limit: -60 Amplitude: -30 Pulse width: 30	Positive Input expected	Does not compile	Fail
Testing Threshold	Lower rate limit: 60 Upper rate limit: 60 Amplitude: 30 Pulse width: 30 Threshold: 50	No Sense	No Sense	Fail
Testing Threshold	Lower rate limit: 60 Upper rate limit: 60 Amplitude: 30 Pulse width: 30 Threshold: 95	Sense	Sense	Pass

DCM

The DCM was implemented using a Python script and a text file to store login credentials, implemented through the use of Python library Tkinter. The text file's number of lines is checked to assure there are no more than 10 users registered at a time. The username and password data is stored in the text file on alternating lines (with the usernames on odd numbered lines and passwords on even numbered lines, starting from line 1). Multiple features have been added to improve user-friendliness and usefulness of the product. Firstly, the user can not register a new user with a blank username or password. The user is also not able to register a new user that has the username of a pre-existing user.

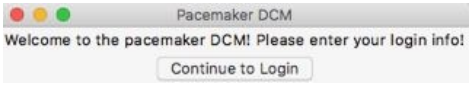


Figure 8: Welcome Screen

Figure 9: Login Screen

Figure 10: Registration Screen

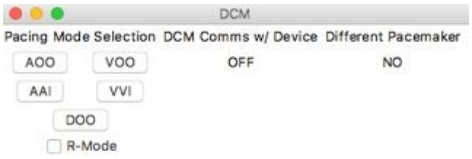


Figure 11: DCM Screen

Figure 12: DOO Mode Inputs

MIS

The modules within our windows.py file are in the table below, the local variables and methods are all given in the table.

Table 20: MIS Table

Module Name	Variables	Methods
windowStart	welcomeLabel continueToLoginButton	
windowLogin	loginButton registrationLabel registrationButton	def matchingUsernameAndPassword (username, password) def login(window)
windowRegister	backButton registrationButton	def register(window)
windowDCM	AOOData VOOData AAIData VVIData DOOData	def parameterInputData(windowName) def getCurrentInputs() def updateInputModeInto() def importSetting() def fillDCMWindow(interface)

Global Variables

AOOData - type array

VOOData - type array

AAIData - type array

VVIData - type array

DOOData - type array

currentMode - identify to simulink which mode to use, the DCM controls this

MID

windowStart

Welcome screen to allow user to access either login or registration window.

windowLogin

Opens login window, takes username and password and verifies user.

def matchingUsernameAndPassword(username, password):

    Check that the username and password are valid and belong to a user in the text file database.

def login(window):

    Given the user information, if input matches user then open the DCM window.

windowRegister

Opens registration window, takes username and password to create user in text file.

def register(window):

    Write username and password data to text file, if it is within the guidelines for a valid username and password. (i.e. password must be at least 6 digits)

windowDCM

Directs user to control window that allows the user to modify the parameters of the pacemaker.

loginAndRegistrationWindowsFunctionality

Parent class to hold all login and registration functionalities. Additionally holds the login and registration classes.

createThread(Self, usernameEntry, passwordEntry):

- Initializes daemon thread for constantly checking username and password.

setCurrentUsername(self, username):

- Sets the username for the whole class to the one that is currently entered into the entry box.

setCurrentPassword(self, password):

- Sets the password for the whole class to the one one that is currently entered into the entry box.

updateUsernameAndPassword(self, usernameEntry, passwordEntry):

- Updates the username and password to the what is written in their respective entry fields.

usernameAndPasswordEntries(self, root):

- Using the Tkinter library, entry areas and their labels are placed on the window according to a grid system. Additionally, the thread for the login is started.

loginWindowsFunctionality

Holds all functionality used for the login window including button layout, credential checking etc

fillLoginWindow(self, lwindow):

- Fills the window with the appropriate buttons and entry areas for logging in.

matchingUsernameAndPassword(self, currentUsername, currentPassword):

- Using the values from setCurrentUsername and setCurrentPassword, iterates through the login credential txt file and compares the currentUsername against all usernames present in the file. If a username is matched with the same username entered by the user, it passes the first conditional and then moves on to check if the password matches that password associated with that username (note that these processes are done concurrently to reduce processing needed). If both are found, then TRUE is returned, however if no matching password or username is found a FALSE is returned. If only one or the other is found, FALSE is returned.

login(self, lwindow):

- Based on whether matchingUsernameAndPassword evaluates to true or false, we will log the user in.

registrationWindowsFunctionality

Holds all functionality used for the registering window including writing to the registered user file, making sure registrations are done properly and alerting the user if there is an issue etc.

fillRegistrationWindow(self, rwindow):

- Fills the window with the appropriate buttons and entry areas for registering a new user

validLogin(usernameOrPassword):

- Iterates through either username or password (depends which is input) and checks if all characters that have been entered are valid characters. If the characters used are invalid, false is returned. Otherwise true is returned.

userAlreadyExists(self):

- Iterates through the txt file of all existing login credentials and compares all of the usernames to the current username trying to be registered. If the same username is found, userAlreadyExists is set to True, if the username is unique, userAlreadyExists sets to false. This value is then returned.

writeToSheet(self, rwindow):

- Using an if statement and several elif statements, invalid registration attempts create a display message showing the user their error. For example, if only a space is entered as a username, the if statement checking that will evaluate and a message explaining the error (no spaces, pre-existing username, password too short, etc) will be shown and an unsuccessful registration message will be displayed.
- If at the end, all invalid username and password checks have passed, the successful set of username and password will be written on a new line to the txt file containing all the other login credentials. Additionally a message will be displayed to notify the user of successful registration.

DCMWindowsFunctionality

Holds all functionality for operating the pacemaker and its parameters.

importSettings():

- Using a second txt file containing values to be imported, reads all the data and stores it in their respective arrays. Based on the windowName, the needed values are read and sent to be used.

def getCurrentInputs():

//Retrieve inputs from window and store them in thread target.

updateInputModeInfo():

- Using an array, stores the to-be-updated values of all used parameters. Then, based on the mode it is in, it updates the parameters to their corresponding value in the array.

parameterInputData(windowName):

- Initially, the mode is read using if statements based on the window name. Then based on which mode of operation we are in, parameters and parameter entry boxes are displayed. Which parameters are changeable entirely depends on which mode was chosen from the initial mode-choosing screen. (Ex. cannot change atrial parameters in ventricle mode, must be the appropriate type and within range) This method has the ability to update the data for the given mode that it is on, and in turn update the global variables for the data and the text file that stores these.

printLists():

- Prints all the mode settings into the console for debugging purposes

fillDCMWindow(interface):

- Fills the window with the appropriate buttons and entry areas for updating and using the DCM and all of its modes.

newPacemaker():

- Using an if statement checks if there is a new pacemaker being used, if there is a YES is displayed, if nothing is detected, NO is displayed. This mode would be fully implemented if we were to receive serial communications from Simulink to the DCM.

There are no private functions in the modules as we implemented our code in Python, while this may make our code less secure, we tried to minimize the risks by using local variables as much as possible as safety is a concern.

Serial Communication Protocol

Assignment 2

Serial communication starts off with a port check to find the serial device. If the device is found, then the port is opened and serial communication is initialized. A boolean variable is additionally set as a flag for serial communication (whether it is or isn't serially communicating) called failed. If failed is set to TRUE, that means the serial communication has failed and the process stops. If failed is set to FALSE, then serial communication is allowed and the values are transmitted as floats via byte packing.

The byte packing protocol is as follows:

AOO:

[lowerRateLimitInput, upperRateLimitInput, atrialAmplitudeInput, atrialPulseWidthInput, ARPInput, **1**, 0, 0, 0, r_mode]

VOO:

[lowerRateLimitInput, upperRateLimitInput, ventricularAmplitudeInput, ventricularPulseWidthInput, VRPInput, **2**, 0,0,0,r_mode]

AAI:

[lowerRateLimitInput, upperRateLimitInput, atrialAmplitudeInput, atrialPulseWidthInput, ARPInput, **3**, 0, 0, 0, r_mode]

VVI:

[lowerRateLimitInput, upperRateLimitInput, ventricularAmplitudeInput, ventricularPulseWidthInput, VRPInput, **4**, 0, 0, 0, r_mode]

DOO:

[lowerRateLimitInput, upperRateLimitInput, atrialAmplitudeInput, atrialPulseWidthInput, ventricularAmplitudeInput, ventricularPulseWidthInput, VRPInput, ARPInput, r_mode]

The digit sent from 1-5 is to indicate the respective mode:

- 1 -- AOO
- 2 -- VOO
- 3 -- AAI

- 4 -- VVI
- 5 -- DOO

A final float, stored as r_mode, is transmitted as either 1.0 or 0.0, representing if rate adaptive mode is enabled.

The ‘0’s in A and V modes are to allow atrial and ventricular amplitude/pulse-width to fall in the same respective array position for their respective mode, allowing easier processing on the simulink side.

Requirements Changes

Assignment 1

Establish login and registration, limited to 10 users. Print a message to user when login or registration fails. The interface should provide a method to select the different modes, AOO, VOO, AAI, VVI. Show mode specific inputs.

Assignment 2

New requirements for the DCM will include adding additional mode controls to DCM window for AOOR, VOOR, AAIR, VVIR, DOOR. Also the transmission of data to tell system to enter rate adaptive mode. Alert the user when changes are sensed by the accelerometer. Design will notify user when device is connected and communicating with the DCM. The device must also notify the user when connection is lost with the device.

Design Changes

Assignment 1

In assignment 1 we addressed some design interface issues by making unnecessary parameters unchangeable. This was done by disabling them which greys out the input field (e.g. ventricle parameters in atrial modes etc.). Some helpful display messages were also added after actions are performed (e.g.

showing “Login successful” or “Registration unsuccessful”). Lastly, we also made the display windows larger for ease of use and readability.

Assignment 2

In assignment 2, some likely design changes for the DCM would include a smoother user interface, and possibly additional buttons for additional rate adaptive modes. Additionally, the way we package data will change as it will be sending more than 1 byte of data to allow for more accurate inputs, and accommodate the new mode parameters. Lastly, possibly adding support for other languages would also be beneficial to expand our user base. We used threading to access data in the background, for the control inputs.

Bonus

Assignment 1

The bonus in assignment 1 uses a push button to inhibit a pace. It mimics a natural pace by viewing the push of a button as a pace of the heart. This causes the Pacemaker to wait and only pace after the escape interval has passed. The button was implemented inside the subsystem used for hardware hiding. By connecting it to an output, and connecting that output as an input to the LED chart, we were able to use the button as a condition when transitioning between the Charge and Natural_Pace states. In other words, the button input was added to our state transition, via the OR operator. This ensured that upon button press, the pacemaker transitions as if it had detected a stimulus.

Testing

Table 21: Testing DCM Code

Purpose	Input	Expected Output	Actual Output	Result (Pass/Fail)
Login First time	Username: user123 Password: test123	Not Registered	Error message printed	Pass
Register First time	Username: user123 Password: test123	Direct to Pace Mode selection page	Direct to Pace Mode selection page	Pass
No inputs for Login	Username: “” Password: “”	Incorrect username/password	Does not login	Pass
Incorrect inputs for Login	Username: a Password: a	“Incorrect login credentials” pop-up	Notifies User	Pass
No input for registration	Username: Test1234 Password: “”	Password must be at least 6 characters long	Asks user to put a longer password	Pass
AOO mode selector Bad lower and upper ranges	Lower rate limit: 500 Upper rate limit: 400 Amplitude: 100 Pulse width: 30	Incorrect range	Displays error message	Pass
VOO mode selector Text input for lower and upper bound	Lower rate limit: “kndk” Upper rate limit: “kmcd” Amplitude: 100	Incorrect type	Error	Pass

	Pulse width: 30			
AAI No input for lower and upper bound	Lower rate limit: “” Upper rate limit: “” Amplitude: 100 Pulse width: 30	Missing inputs	Error	Pass
Negative Amplitude	Lower rate limit: 40 Upper rate limit: 50 Amplitude: -100 Pulse width: 30	Error: cannot have negative amplitude	Error	Pass
Negative pulse width	Lower rate limit: 40 Upper rate limit: 50 Amplitude: 100 Pulse width: -30	Error: cannot have negative pulse width	Error	Pass
Bad type for Amplitude & pulse width	Lower rate limit: 40 Upper rate limit: 50 Amplitude: “cc” Pulse width: “cc”	Error: incorrect type	Error	Pass
No input for Amplitude/pulse width	Lower rate limit: 40 Upper rate limit: 50 Amplitude: “” Pulse width: “”	Error: no input	Error	Pass

These tests cover many of the edge and incorrect input cases and show that our program, while it errors out with some inputs, performs the standardized functions correctly.

Design Principles

In our design for both the DCM and Simulink Model, we went back and modified the entire implementation structure based on the feedback from assignment 1. In assignment 1, our DCM was structured in such a way that the code was all in one file. This was a mistake as it meant our code had low modularity, resulting in less flexibility and variety in use. By taking our complex DCM system and separating it into simpler modules in the form of classes, it allows us to deal with the pieces one at a time which is an application of separation of concerns. This is also beneficial not only for debugging, but is also fundamentally more sound and easier to understand.

Additionally, our design also employs the use of information hiding. Information hiding is a concept similar to encapsulation, it hides the sensitive information from users and only shows the necessary information. An added benefit of it is that it also allows us to separate the code so that we can make changes in one portion without risking it affecting others. In our DCM, we employed information hiding through the use of private classes, and in the Simulink Model, it's through the use of a subsystem holding all the input pins and buttons. In assignment 1, we separated the Simulink model into two components, an FSM and a sub-system for the board inputs. There were three states, with two for charging and discharging and the last for sensing a pulse from the heart. The pin assignments for the switches responsible for pacing/charging the heart/capacitors are placed within the subsystem. This encapsulation is a form of hardware hiding as it allows us to simplify the complex design, as well as allow us to make changes and add components for assignment 2 without affecting assignment 1

Our simulink model employs high cohesion and low coupling. Hardly any functions are called, as the stateflow is handled mainly by the transition conditions. When rate adaptive mode is implemented, the cohesion increases as states must be created to process the acceleration values, however the only affect this has on other states is through modifying the global BPM value. We could reduce the complexity of code by simplifying transitions, and having defined states to handle user input.

Lastly, even though we tried to go for high cohesion and low coupling, our current DCM model is highly cohesive but also has moderately high coupling. We were able to improve on the cohesion and coupling that we had in assignment 1 by re-organizing our DCM from groups of nested methods in one python file

to using a more intuitive class system spread across multiple files. Doing this also improved the readability and understandability of the code. The high cohesion, however, comes from the fact that the components in our class modules are closely related to one another, but there is also high coupling as our classes and functions use other classes and functions. To try and limit this problem as much as possible, we used local variables where applicable, and for the most part only updated global variables after the local variables were done being changed to limit the possibilities of changing variables accidentally.

DCM Code

https://github.com/gohabs19/Group11PacemakerDCM/tree/master

References

"Savitzky-Golay Filtering." *Savitzky-Golay Filtering - MATLAB*, www.mathworks.com/help/signal/ref/sgolayfilt.html.

Schafer, Ronald W. *What Is a Savitzky-Golay Filter?* inst.cs.berkeley.edu/~ee123/fa12/docs/SGFilter.pdf.

Meyer, Guy. *Pacemaker Microcontroller Shield*. 2019, pp. 1–20, *Pacemaker Microcontroller Shield*.

Scientific, Boston. *PACEMAKER System Specification*. Boston Scientific, 2007, pp. 1–35, *PACEMAKER System Specification*.

"Pacing Codes and Modes Concept." 2019.