Assignment 2 – Pacemaker Design

SOFTWARE DEVELOPMENT 3K04

GROUP 14: PIKA

Table of Contents

Table of Tables and Figures

Group Members

Name	Student Numbers	McMaster Emails
Ahmed Afifi	400066042	afifia1@mcmaster.ca
Mark Danial	400066296	daniam1@mcmaster.ca
Abdulrahman Elgendy	400051947	elgendya@mcmaster.ca
Mina Ghaly	400052424	ghalym1@mcmaster.ca
Mohamed Mahmoud	400078830	mahmom10@mcmaster.ca
Omar Mouftah	400080124	mouftaho@mcmaster.ca

Introduction

When the heart's natural pacemaker is not working properly, the heart may beat too fast, too slow or in an irregular pattern. Therefore, a pacemaker would be a great solution to restore the hearts natural rhythm, as it monitors the operation of the heart and detects malfunctions quickly. Pacemakers are connected to the heart through wires called leads in which electrical pulses are rapidly sent to help stimulate the heart when needed. In Assignment 1 we were asked to develop an open-loop state flow that implemented both VOO and AOO functionality. Therefore, our design was based upon the idea of how to pulse the ventricle/atrium at a given rate while taking pulse characteristics and rate characteristics into account.

Building on Assignment 1, this assignment was designed to implement not only VOO and AOO, but VVI and AAI as well as the rate adaptive functionality of all 4 mentioned modes. Meaning, the design had to account for sensing circuits and accelerometer inputs. Moreover, to achieve proper functionality of the pacemaker and dynamic switching between modes, serial communication had to be implemented. This is to connect the DCM to the pacemaker and observe how it behaves to different inputs while dynamically switching modes. As discussed in the *"Bonus"* section of Assignment 1, a pushbutton was used as an abstraction of the sensing circuit. For this assignment, that pushbutton was substituted for the proper sensing circuit, this will be discussed in *"Design Decisions"*.

Explanation of Pacing

Figure B.2 visually shows how artificial pacing occurs in the heart. The pacing capacitor is responsible for the actual pacing of the heart while the blocking capacitor collects the excess charge that would have been accumulated in the heart. The blocking capacitor would then slowly discharge through the heart to produce a net zero current flow. The pacing process begins by firstly connecting switch S2 to ground and then closing switch S1. This leads to the charging of the pacing capacitor and discharging of the blocking capacitor. The second step would be to discharge the pacing capacitor into the heart and allow the excess charge to charge up the blocking capacitor. This is achieved by firstly opening S1 to disconnect the pacing capacitor from the PWM source and then moving S2 such that the PG lead and pacing capacitor are connected to each other. Eventually, this pacing process will keep happening recursively, and a pulse will be released to the heart every set interval (steady pulse). This interval can be calculated using the following formula:

Equation 1: Pacing Interval Formula

$$\text{Interval (ms)} = 60{,}000 \,/\, \text{rate (ppm)}$$

Explanation of Sensing

The sensing circuit in Figure B.3 is activated by closing the switch labelled "FRONTEND_CTRL". Afterwards, a reference PWM needs to be inputted and that will act as the comparator threshold. The comparator then compares the rectified signal from a natural heartbeat to the PWM threshold input. If "rectified signal > threshold", then the sensing circuit outputs a Boolean TRUE which means that a natural beat was sensed. For VOO and AOO, the pacemaker is only concerned with the rate at which it pulses the

heart, completely omitting any natural pulsing signals the heart may produce. VVI and AAI, however, must inhibit an artificial pace whenever the heart naturally beats. As can be seen in Figure 1 (below), whenever a sensed signal (V_s) occurs in the ventricle, the pacemaker should not pulse (V_p). The same is true for similar atrial activities in AAI. In addition to inhibiting the artificial pulse, VVI and AAI must also include a refractory period (VRP/ARP). A period of time occurring after a ventricular or atrial event during which the sensing system should not trigger nor inhibit a pace. Meaning, the sensing circuit must be ignored throughout VRP before reacting to any more sensing circuit outputs.

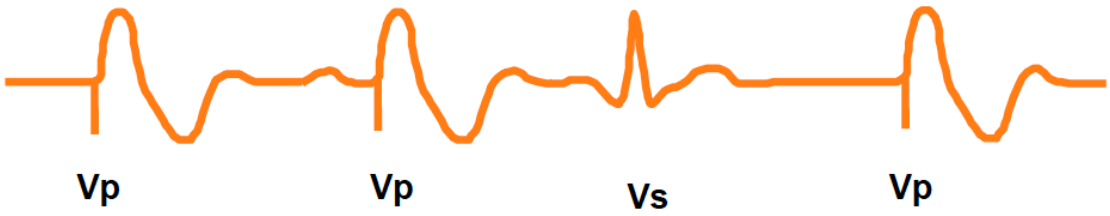

Figure 1: VVI Timing Cycle

Requirements Likely to Change

Going from Assignment 2 to Assignment 3 there are a few changes in the requirements for the deliverables expected. The second assignment focused on creating the stateflow charts and implementation of VOO, AOO, VVI, and AAI while also making sure the rate adaptive functionality is implemented for all 4 mentioned modes. The serial communication was also established to communicate between the DCM and Simulink for the pacing modes and for the user to dynamically switch between modes without restarting the device.

Assignment 3 requires the implementation of DDD and DDDR. DDD indicates that the system should be able to pulse, sense, and implement a tracked response to sensing in both chambers (atrium and ventricle). This means an AV delay is necessary since in practical terms, both chambers don't naturally pulse simultaneously, even in a healthy heart. A tracked response to sensing means that after an atrial sense, there should be a tracked ventricular pulse after a specified AV delay, unless there was a sensed ventricular heartbeat beforehand.

To implement these modes, many more programmable parameters will need to be introduced in the System Inputs subsystem, ensuring that serial communication remains functional. For example, in VVI and AAI, the refractory periods (VRP and ARP, respectively) were used to ignore any sensed activities following a chamber's event. In DDD, PVARP must be implemented along with VRP and ARP. PVARP (Post Ventricular Atrial Refractory Period) is the period of time following a ventricular event in which an atrial cardiac event should not inhibit an atrial pace nor trigger a ventricular pace.

Figure 2 (below) shows the timing for dual-chamber functionality. The total time taken to pulse the ventricle (V_p) is the sum of the AV delay and the time between V_p and the atrial pulse (A_p).

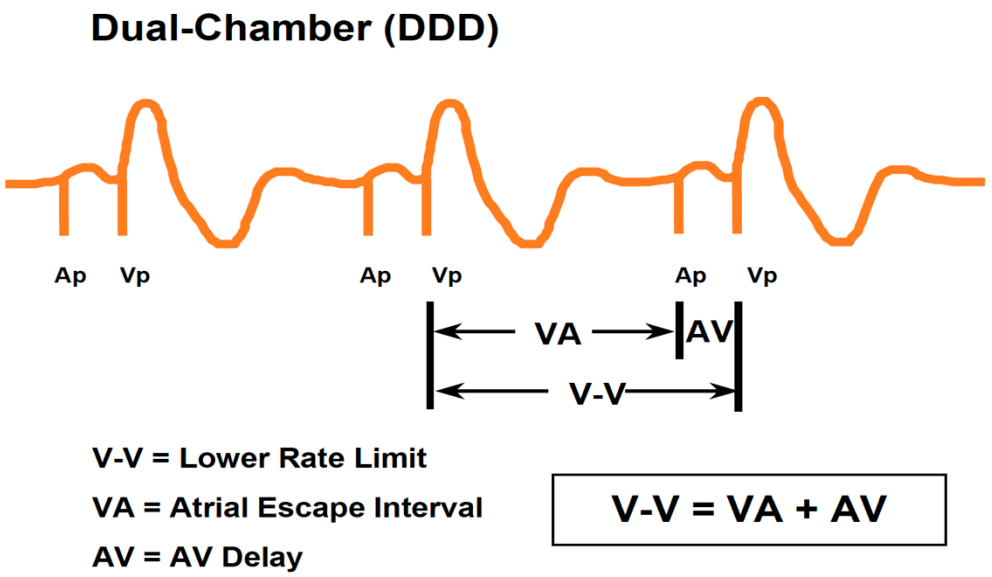

Figure 2: Dual Chamber Timing

The timing cycle above changes depending on whether the system is sensing the atrium and pulsing the ventricle, or vice versa. The system could also be sensing both chambers should the heart be naturally pacing. Such assumptions along with the programmable parameters will need to be taken into consideration for Assignment 3.

Design Decisions Likely to Change

As mentioned in the previous section, DDD requires many additional programmable parameters but since our current Simulink model has every mode separated into its own triggered subsystem, it would be relatively easy to add to the Simulink model without affecting other modes.

Having said that, some changes will need to occur. The System Inputs subsystem will have all the new programmable parameters added to it, making sure they can be accessed and changed via serial communication. Additionally, the Pacing subsystem will require more cases in the switch case block discussed above in the *"Pacing Subsystem"* section which currently has 8 cases, 1 for each mode, it is certain that 2 more cases will be added. Furthermore, since both modes will not only share outputs with each other but with the existing 8 modes, more MATLAB functions will need to be added to decide which pins will be accessed depending on which mode is running, something that will be controlled through the DCM.

Design Decisions

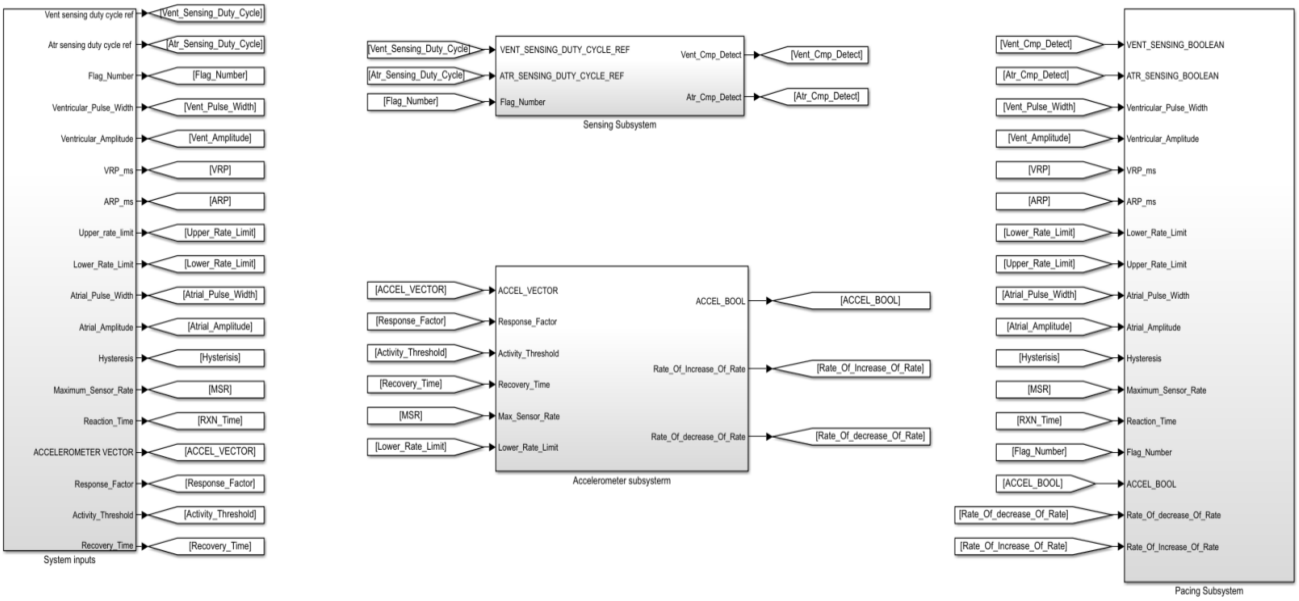

Figure 3: Simulink Model

Design Overview

The Simulink model is split into numerous subsystems which are meant to organize the model, thereby introducing a structured method of navigating the system. The main subsystems are outlined above in Figure 2. The main design includes 4 subsystems; System Inputs, Sensing Subsystem, Pacing Subsystem, and Accelerometer Subsystem. System Inputs includes all inputs used in any of the pacing modes. It also includes the Serial Communication subsystem, which is then connected to all programmable parameters needed. These inputs are then connected to any of the other 3 subsystems based on any computations that need to be done before being linked with their respective modes. An example of this is the Response factor which is used in the Accelerometer Subsystem to help compute the rate of change of the current pacing rate. The output of the accelerometer is then connected to the pacing subsystem. As can be seen from the figure above, one of our design decisions was to use Goto and from transition blocks to eliminate the mess that is associated with using arrows. These blocks work the same as the arrows but provide a much more organized look that is easy to follow. Within the Sensing and Accelerometer subsystems, decisions were made on the appropriate programmable parameters and how to use them within their respective modes. The Pacing subsystem included all 8 modes, each in its own triggered subsystem. Each mode has its respective flag number, and this allows us to change modes dynamically through the DCM.

Sensing Subsystem

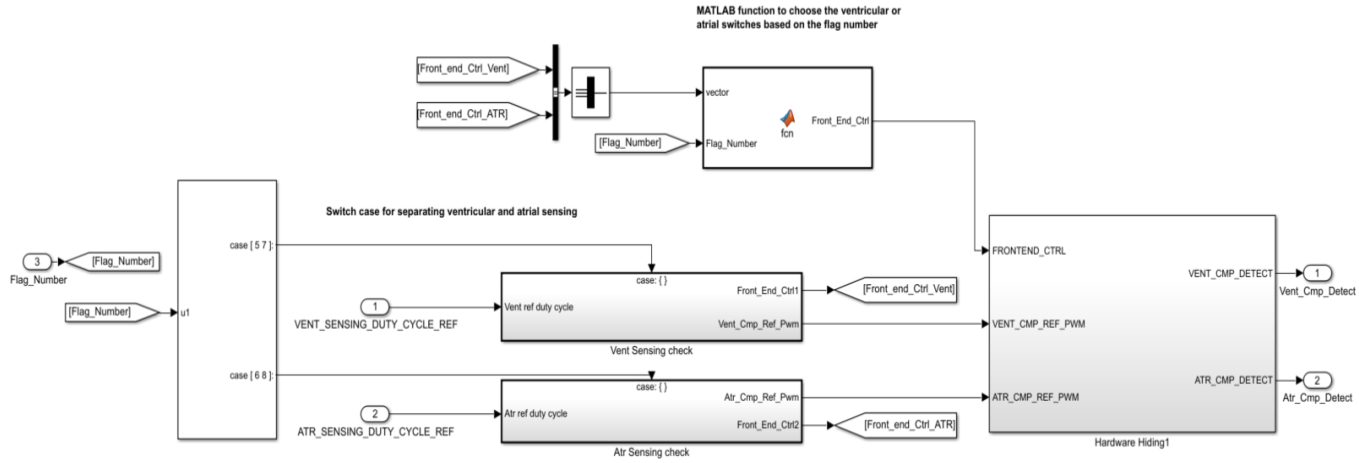

Figure 4: Sensing Subsystem.

Figure 2 displays the sensing subsystem, which determines whether the heart was naturally pacing based on the Labview testing environment. The sensing subsystem takes in three inputs: Ventricle reference duty cycle, Atrial reference duty cycle, and a flag number which represents the required mode. On the left of the figure, there is a switch case block that takes the flag number as an input and decides whether to trigger the Ventricle Sensing check or Atrial Sensing check based of the flag number. After it jumps to either the Vent/Atr sensing check subsystems, a state flow chart is triggered that activates the sensing circuit by closing the “FRONTEND_CTRL” switch and setting a reference PWM voltage that will act as the threshold. This can be seen below in figures 4 and 5.

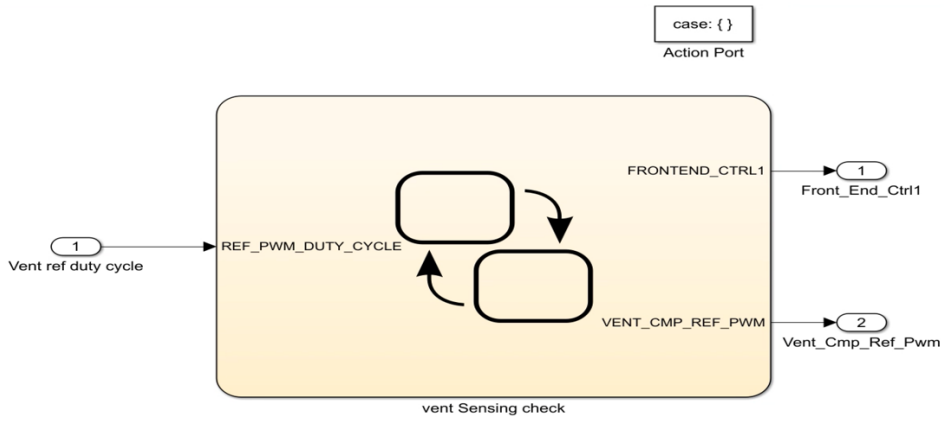

Figure 5: Ventricular Sensing Check

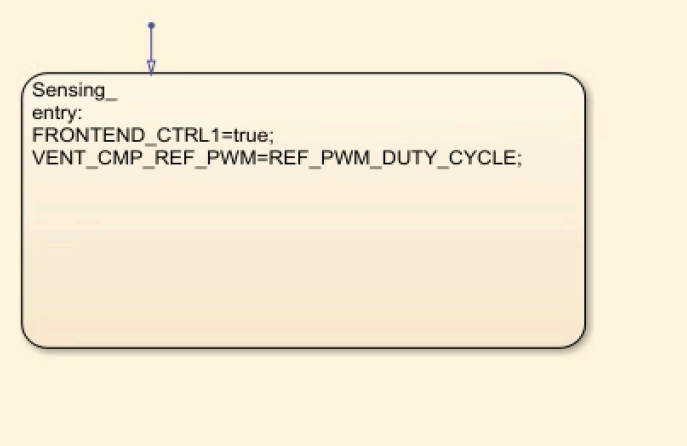

Figure 6: Setting Reference PWM Cycle

One of our design decisions was to use switch cases to trigger/activate the sensing circuit. The sensing circuit is only required for four modes (VVI, AAI, AAIR, and VVIR), and the switch case block only supports those 4 modes. Therefore, our design activates the sensing circuit only when needed and this helps avoid unnecessary computations. Some of the issues that we faced was the fact that you could only define a specific hardware pin once in your Simulink model, and you also cannot connect multiple signals to one hardware pin at the same time or else you would run into multiple errors. Both the Vent/Atr sensing subsystems shared the same hardware pin labelled "FRONTEND_CTRL". Therefore, to get rid of this issue, one of our design decisions was to create a MATLAB function that decides whether to connect the "FRONTEND_CTRL" hardware pin to the Atrial or Ventricle Sensing subsystems based of the flag number that's inputted through the DCM. This MATLAB function takes in a vector containing the Front-End Ctrl pin from the Ventricle and Atrial sensing subsystems, and it assigns the hardware pin to one of them based on the flag number. For example, if the flag number was 5, which corresponds to VVI, the MATLAB function sets the hardware FRONTEND_CTRL pin to Front_end_Ctrl_Vent from the Ventricle sensing subsystem. The opposite is true for case 6 (AAI).

```
function Front_End_Ctrl = fcn(vector,Flag_Number)

switch Flag_Number
    case 5
        Front_End_Ctrl=vector(1);
    case 6
        Front_End_Ctrl=vector(2);
    case 7
        Front_End_Ctrl=vector(1);
    case 8
        Front_End_Ctrl=vector(2);
    otherwise
        Front_End_Ctrl=false;
end
```

Figure 7: MATLAB Switch-Case Function

After closing the “FRONTEND_CTRL” switch and assigning a reference PWM, the sensing circuit compares the rectified signal from a natural heartbeat to your reference PWM, and it assigns a Boolean to a pin on the board that you can later “digital read” into your Simulink model as seen in our hardware hiding below in Figure 7.

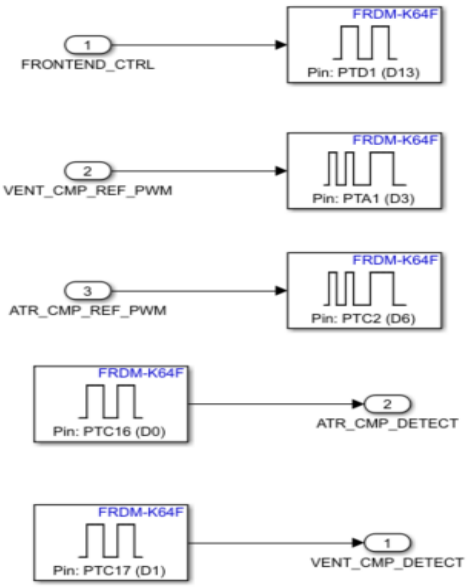

Figure 8: Sensing Hardware Hiding

Accelerometer Subsystem

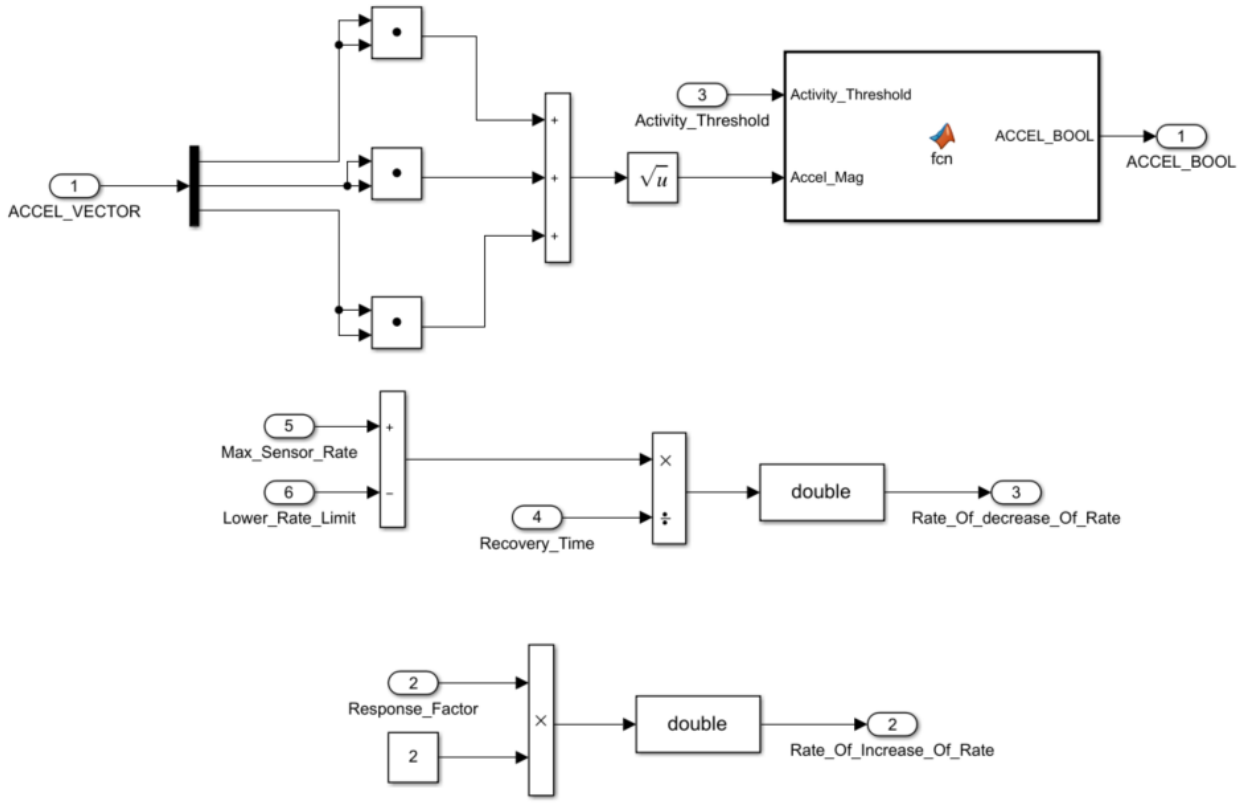

Figure 9: Accelerometer Subsystem

The accelerometer subsystem takes in 6 inputs: Accel_Vector, Response_Factor, Activity_Threshold, Recovery_Time, Max_Sensor_Rate, and Lower_Rate_Limit. Firstly, the acceleration vector from the K64F on-board accelerometer was split into its three components (x, y, and z). After that, the different acceleration values were squared, summed, and square-rooted to obtain its magnitude ($\sqrt{u}$). A MATLAB function was then used to assign a Boolean value to ACCEL_BOOL. If the magnitude of acceleration was greater than the activity threshold, then the function would assign TRUE to the ACCEL_BOOL and FALSE otherwise. One of our design decisions was to analyse the total magnitude of the acceleration rather than analyse the acceleration of the three different axes individually. This was done to eliminate any noise picked up from small movements. The ACCEL_BOOL is then sent to the pacing subsystem where the current mode checks whether the acceleration Boolean is true or not. If it's true, then the pacing mode would increase the pacing rate. Therefore, our accelerometer subsystem would calculate the "Rate_Of_Increase_Of_Rate" using the response factor as shown in the figure above. We know that a response factor of one leads to a rate of increase of rate of 2BPM/s (this information was obtained from a document called "rate adaptivity explained" provided to us on Avenue to Learn). Therefore, the rate of increase of rate could be modelled as (Response_Factor * 2), and this rate of increase is sent back to the current pacing mode to help increase the pacing rate. When ACCEL_BOOL finally becomes false, the pacing mode would decrease the current pacing rate to the lower rate limit (which was the initial pacing rate). To do so, the accelerometer subsystem would need to calculate the

rate of decrease of the rate. Since the rate of decrease of the rate can be modelled as a deceleration, we derived our equation from the following formula:

Equation 2: Deceleration Equation

$$deceleration = \frac{V_{initial} - V_{final}}{time}$$

Therefore, our formula was modelled as:

Equation 3: Rate of Decrease of Rate Formula

$$\frac{Max\ Sensor\ Rate - Lower\ Rate\ Limit}{Recovery\ time}$$

One of our design decisions was to have the rate level go from the Lower Rate Limit to the Max Sensor Rate gradually by doing the calculations mentioned above. This is to ensure that the heartrate did not jump from 60 BPM to 120 BPM in 1 step/1 beat as that would not be realistic. Another design decision was to give the doctor/user the ability to change the rate of decrease/increase of rate through the DCM by having variables like Max Sensor Rate, Lower Rate Limit, Response Factor, and Activity Threshold as programmable parameters rather than defining them as constants in our model. This gives the user the ability to change these parameters dynamically through the DCM, allowing the user to account for the different requirements that different people need.

Pacing Subsystem

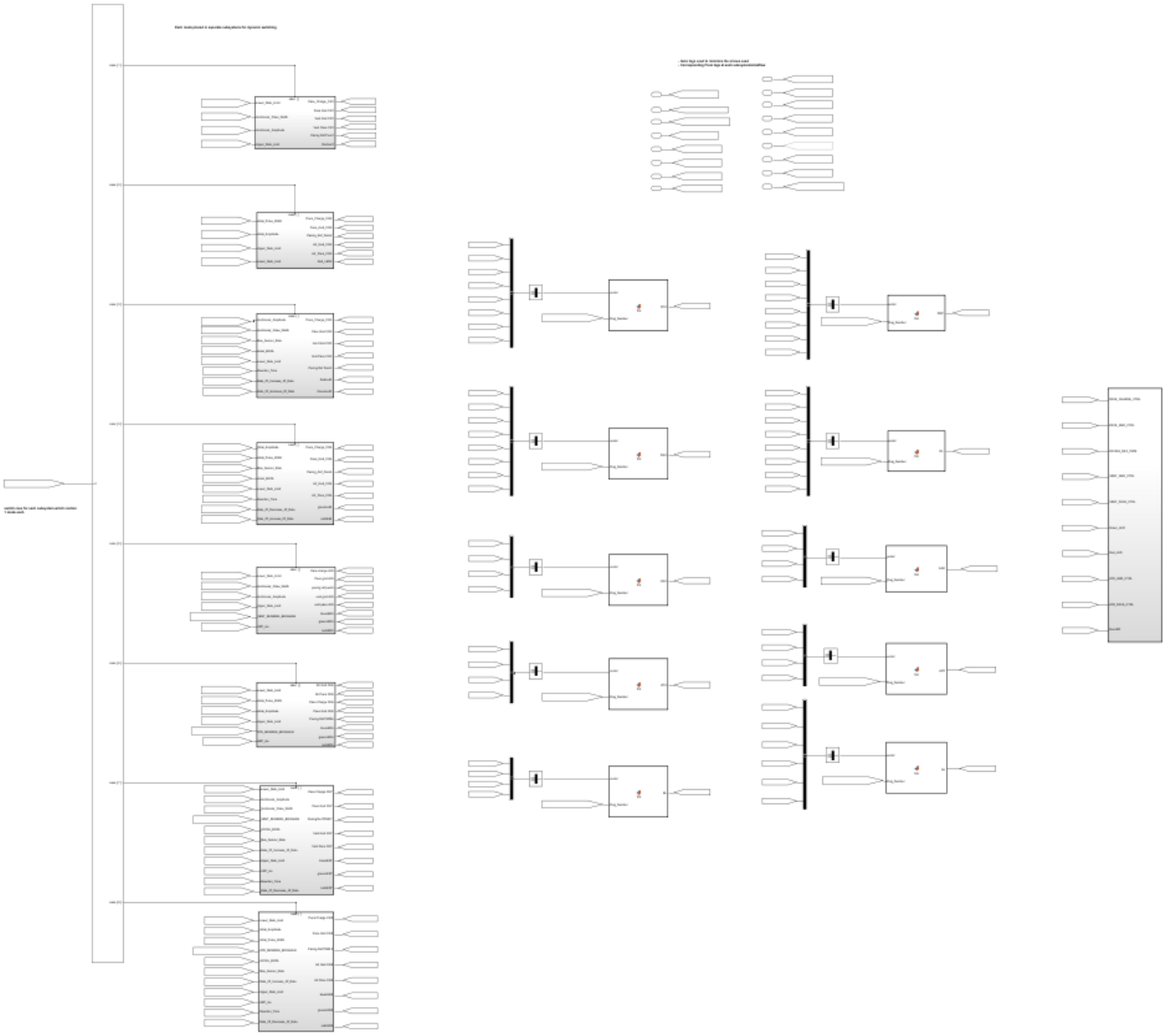

Figure 10: Pacing Subsystem

Figure 9 shown above is the main pacing subsystem where all the pacing modes are placed. Each mode was placed into its own triggered subsystem to make it easier to transition dynamically between modes based on the flag number. Every triggered subsystem has a flow chart that represents its respective mode. Figure 10 (below) shows this more clearly.

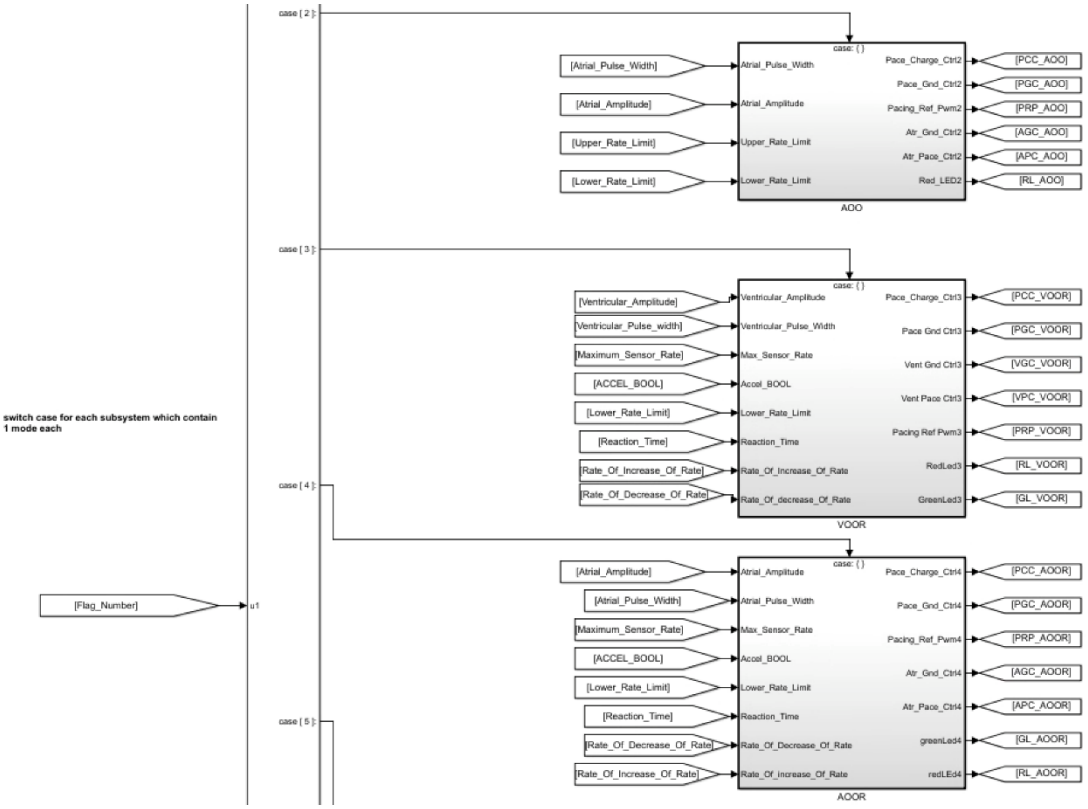

Figure 11: Pacemaker Mode Subsystem

To the left of Figure 10 is a switch case block that takes in the flag number as an input and then triggers the mode based on its respective flag number. Therefore, changing the flag number through the DCM dynamically changes the current pacing mode. One of our design decisions was to put every mode in its in own subsystem instead of having all modes in one flow chart. This prevents a lot of unnecessary computations because the only check needed to trigger a certain mode is the flag number. If we were to put all the modes in one flow chart, we would have to make multiple checks to dynamically change between modes. It also helps keep our model clean and organized.

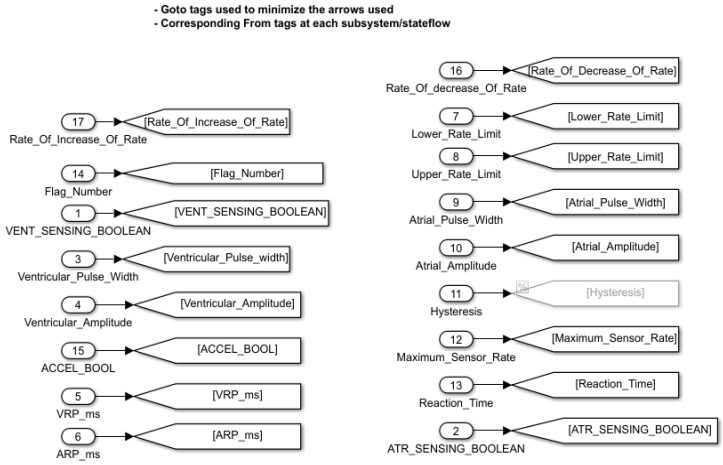

Figure 12: Goto-From Blocks

The programmable parameters are taken as inputs at the top of Figure 9 above. They are then distributed within the pacing subsystem using Goto, and from blocks to keep the model clean and readable.

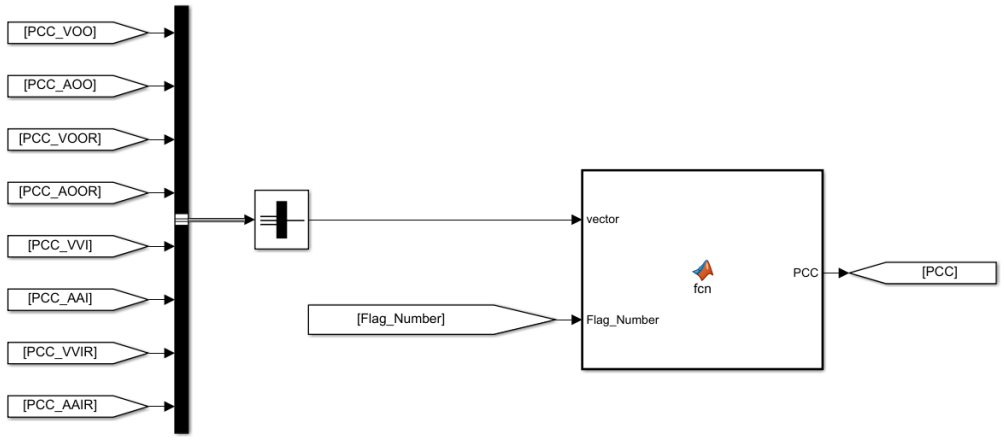

Figure 13: Assigning hardware pin

One of the issues we faced while implementing the modes in 8 different subsystems is the fact that we were only able to define the hardware pins once, but all the modes had some pins in common. Therefore, one of our design solutions was to write a MATLAB function that takes in the flag number and a vector containing the shared pin from all 8 modes. The function would then assign the hardware pin to one of the subsystems based on the flag number. For example, in the above figure, the Pace Charge Ctrl (PCC) pin is shared across all eight modes. If the flag number was 3, which corresponds to VOOR, the MATLAB function sets the hardware Pace Charge Ctrl pin to the Pace Charge Ctrl pin from the VOOR subsystem. The function is shown in the figure below.

```
function PCC = fcn(vector,Flag_Number)
PCC = vector(Flag_Number);
```

Figure 14: Parameter Assigning Function

However, when a certain pin Is not shared by all modes, then writing this function gets a little bit tricky. For example, Vent Ground Ctrl (VGC) is only used/shared by the four ventricle modes.

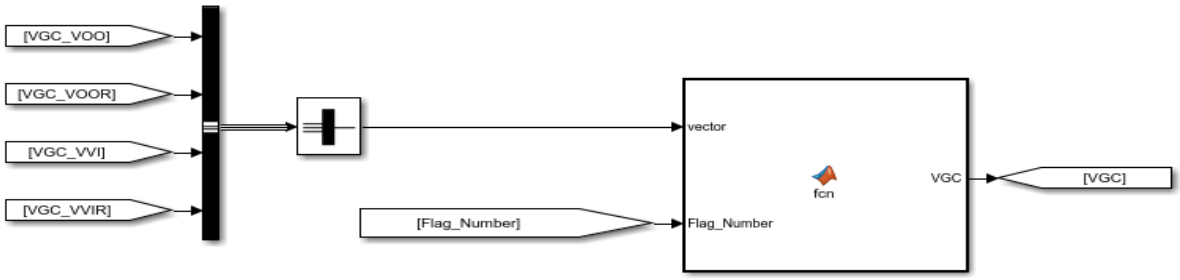

Figure 15: Assigning hardware pin

Therefore, we decided to implement a switch case function that takes the flag number as the case conditions, and then assigns the Vent Pace Ctrl hardware pin to its respective mode based on the flag number. In the function below, The flag numbers of 1,3,5,7 represented VOO, VOOR, VVI, VVIR respectively.

```
function VPC = fcn(vector,Flag_Number)

switch Flag_Number
    case 1
        VPC=vector(1);
    case 3
        VPC=vector(2);
    case 5
        VPC=vector(3);
    case 7
        VPC=vector(4);
    otherwise
        VPC=false;
end
```

Figure 16: MATLAB Switch-Case Function

There was a total of 10 pins that were shared between the modes. Therefore, we implemented ten MATLAB functions to account for that as seen in figure 9 above. One of our design decisions was to use a switch case whenever we needed a MATLAB function. This is justifiable, as the computation is faster than having multiple if statements.

Hardware Hiding

On the right side of Figure 8 is the Hardware Hiding subsystem. This is used in case the hardware changes. If such a situation was to occur, only this subsystem would need to change, leaving nothing to be changed in the mode subsystems and state flows. Figure 15 is a representation of the hardware hiding subsystem, which also shows the interior of the subsystem.

Figure 17: Pacing Hardware Hiding (left) & Interior of Hardware Hiding (right)

Pacing Modes

VOO/AOO

Inside the VOO subsystem is a stateflow chart, which implements the ventricular pacing functionality. As can be seen in Figure 13 (below), the initial state is labelled “Charging_C22_and _Discharging_C21” where C22 and C21 are the pacing capacitor and blocking capacitor, respectively.

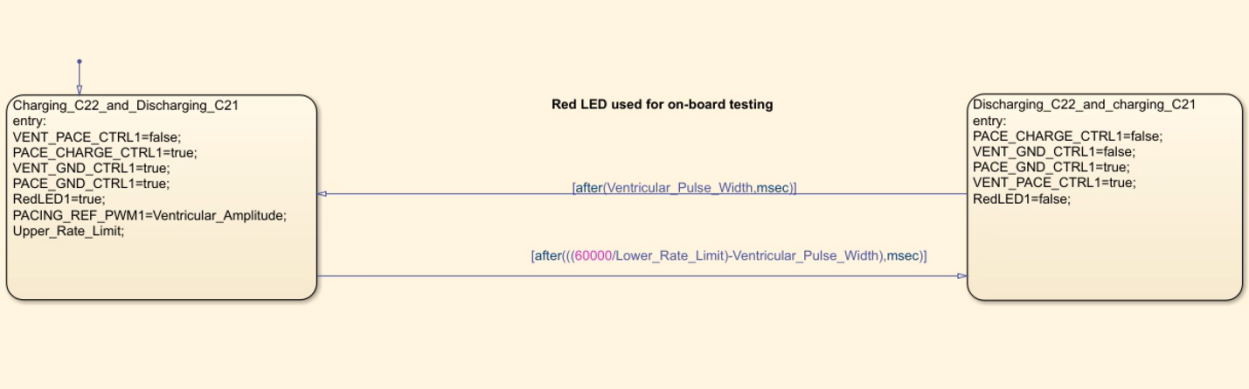

Figure 18: VOO Stateflow Chart

This state will be referred to as “state1” for pragmatic purposes. State1 was chosen to be the starting state for safety reasons. Since this state is used for charging rather than pacing, it is the safer option as the starting state as it eliminates the risk of pacing the heart immediately when not needed.

This also allows C22 to properly charge before the first pacing signal is required. As discussed above, calculations were made regarding how long the simulation should stay in each state based on the pulse width. If the rate is set to 60ppm, this means the heart should be paced every second. Therefore, the simulation remains in state1 for 996ms and paces for 4ms (the pulse width) before returning once more to state1, waiting for the next pulse.

Within state1, the system manipulates the switches found in Figure B.1. Firstly, VENT_PACE_CTRL is opened then PACE_CHARGE_CTRL, VENT_GND_CTRL, and PACE_GND_CTRL are all closed. This charges C22 whilst discharging C21. After 996ms, the time the system waits before pacing, the system transitions to "Discharging_C22_and_Charging_C21", which will be referred to as state2, where it opens the PACE_CHARGE_CTRL and VENT_GND_CTRL switches and closes VENT_PACE_CTRL. PACE_GND_CTRL remains closed. After the pacing duration is complete, 4ms, the system returns to the starting state, state1.

The process is similar in the AOO pacing mode with the only change being the names of the variables. Variables and switches for the atrium were used instead of the ventricle. For this assignment, the subsystems were separated so AOO can be found inside the AOO subsystem with identical design decisions but different variable names and hardware pins.

VVI/AAI

In Assignment 1, a pushbutton was used as an abstraction of the sensing circuit. For this assignment, however, the sensing circuit (Figure B.3) was taken into consideration. Below is the stateflow for VVI. The top 2 states are essentially identical to those of Figure 13 (VOO). The only difference being the transition from Charging_C22_and_Discharging_C21 (state1) to Discharging_C22_and_Charging_C21 (state2) is now shorter in length to account for the refractory period (VRP). As mentioned in *"Explanation of Sensing"* the VRP is a period of time following a ventricular event during which the sensing circuit should not trigger nor inhibit a pace.

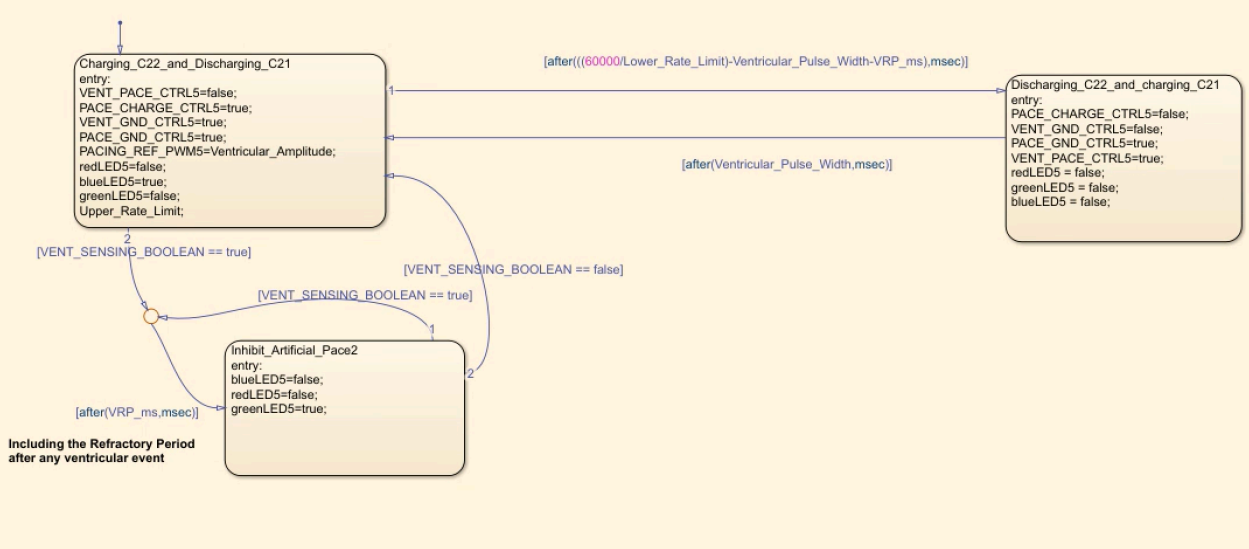

Figure 19: VVI Stateflow Chart

As can be seen, all 3 of the on-board K64F LED's were used for testing. This stateflow takes in VENT_SENSING_BOOLEAN as an input from the sensing subsystem to determine whether the heart is naturally pulsing. While the system is waiting to transition from state1 to state2 (i.e. pulsing), the system is also checking whether VENT_SENSING_BOOLEAN is true. Once it is, the system is guided to a junction, followed by the VRP. It is then directed into the Inhibit_Artificial_Pace2 state (state3), where it will flash the green LED to signal its entry in the inhibition state. When in state3, the system then continues to check if the heart is still naturally pacing, when VENT_SENSING_BOOLEAN turns false, the system will go back to state1 and the process is repeated.

If the system does not sense a natural heartbeat, it will transition to state2 as soon as the transition period is over. To indicate pulsing, the blue LED is turned off since it is naturally on while in state1.

VOOR/AOOR

To implement VOOR, the stateflow takes the same inputs discussed in VOO but with additions that include: Max_Sensor_Rate, Reaction_Time, Rate_Of_Increase_Of_Rate, Rate_Of_Decrease_Of_Rate, and ACCEL_BOOL. As discussed in the *"Accelerometer Subsystem"* section of this document, ACCEL_BOOL is a Boolean value that returns TRUE if the magnitude of the on-board K64F accelerometer vector is greater than the activity threshold and FALSE otherwise. Max_Sensor_Rate and Reaction_Time are programmable parameters from the DCM. The Rate_Of_Increase_Of_Rate and Rate_Of_Decrease_Of_Rate, on the other hand, are derived from the *"Accelerometer Subsystem"* section as well. They are used to increase and decrease the pacing rate to adapt to any activities the person might be doing.

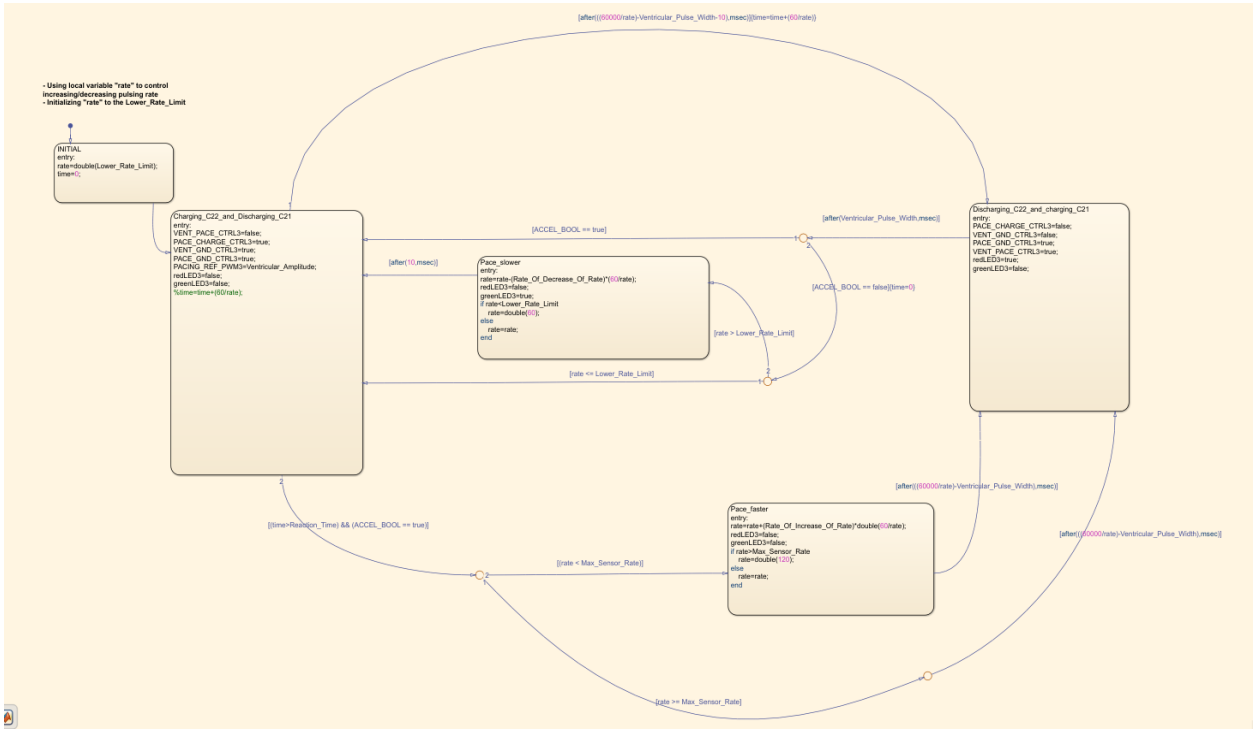

Figure 20: VOOR Stateflow Chart

The INIT state seen in the figure above is only used to set the local variable "rate" to the Lower_Rate_Limit input and set the local variable "time" to zero, thus making it assignable in later states. This was an error that we received and using a local variable eliminated that error. The Charging_C22_and_Discharging_C21 (state 1) and Discharging_C22_and_Charging_C21 (state 2) states are identical to those in the VOO stateflow chart. Green and red LED's were used for testing purposes, where the green LED would flash whenever the system was slowing the pace down and the red LED would flash when the system would pulse. The transition period used to go from state 1 to state 2 is identical to that in the VOO stateflow chart, as well.

If the accelerometer subsystem outputs a TRUE to ACCEL_BOOL (i.e. activity is occurring), then the local variable "time" would need to increase to the Reaction time in order to increase the pace. This is shown in the flow chart by using the equation time=time+(60/rate). However, if the ACCEL_BOOL turn false at any point during the increase in time, then the local variable time is set back to zero. Now, if the time became greater than the reaction time, then the model can go ahead and increase the pace. It then checks whether the current rate is less than or if it is greater than/equal to Max_Sensor_Rate. If it is still less than the maximum sensor rate, it will go to Pace_faster (state 3). In which it will increase the rate, but if the rate is somehow greater than the max sensor rate, it will set it to 120 – the maximum rate – and pulse after the subsequent waiting period. Conversely, if the rate at the junction is greater than or equal to Max_Sensor_Rate, the system will just pulse without increasing the rate any further.

After pulsing and waiting for the pulse width, the system is redirected to another junction where it checks whether the person is still accelerating, if it is it goes to state 1 and repeats the process. If acceleration is no longer true at the junction, the system checks if the rate is greater than or less than/equal to the lower rate limit. If it's greater, that means the pulsing rate is higher than it needs to be and the system goes to Pace_slower (state 4) where it will decrease the rate, but if the rate is somehow less than the lower limit, it will set the rate to Lower_Rate_Limit and go to state 1 and resume the process. If the rate was less than or equal to lower rate limit, the system skips state 4 and goes straight to state 1.

VVIR/AAIR

The stateflow chart for VVIR is shown below (Figure 16). It contains all the same inputs and states as the VVI mode, but it also includes the rate-adaptive programmable parameters discussed in *"VOOR/AOOR"*. VVIR also needed the current rate to be assignable so the INIT state in this case is identical to that of the VOOR stateflow and serves the same purpose. Once again, states 1 and 2 are the same in terms of charging and discharging the pacing and blocking capacitors in the appropriate time and order.

The real difference between this mode and the VOOR mode is that not only does it use data from the accelerometer subsystem, but it also relies on the sensing subsystem. As stated in the *"VVI/AAI"* and *"Sensing Subsystem"* sections, the sensing subsystem outputs a Boolean whenever a natural beat is sensed in its appropriate chamber.

When compiled, the program automatically waits the VRP then is directed to a junction, in which it waits the normal time period before pulsing (state 2). If the heart naturally paces, it will inhibit the next pace (state 3) and go back to state 1 (charging state). However, if acceleration occurs while it is in state 1, the program goes to another junction where it checks the current rate and if it is below or above/equal to the maximum sensor rate. If the rate is less than max sensor rate, it will go to Pace_faster (state 4) and increase the current rate which will then be followed by a pulse state 2. If the rate was greater than or equal to max sensor rate, it would have directly gone to state 2.

After the system pulses at state 2, it checks again whether the acceleration Boolean is still true. If it is true, it goes to state 1, otherwise it checks whether the rate is greater than the lower rate limit which would cause it to go to Pace_slower and then to state 1. If the rate was less than or equal to the lower rate limit, it would not slow the rate down and go straight to state 1.

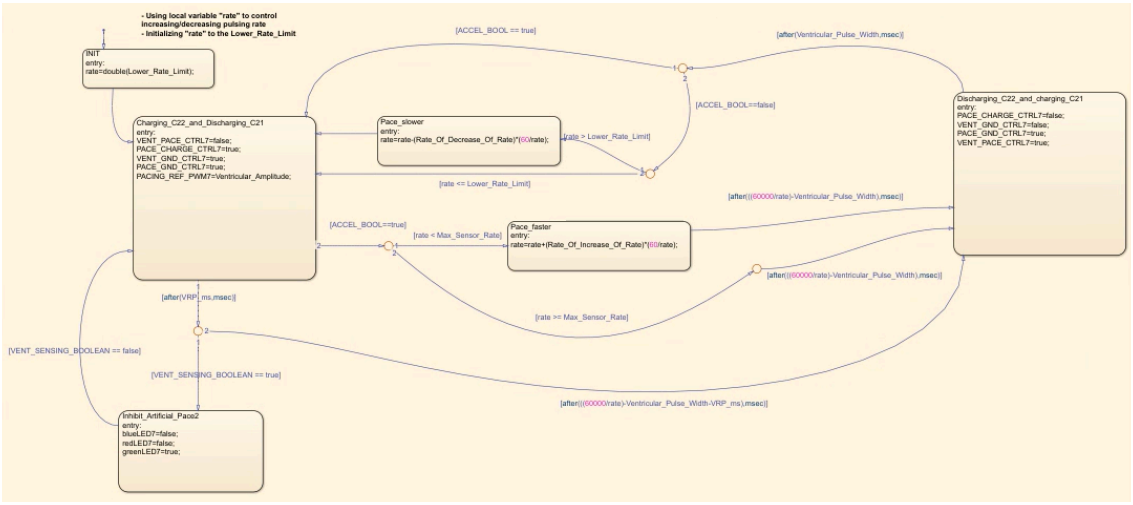

Figure 21: VVIR Stateflow Chart

System Inputs

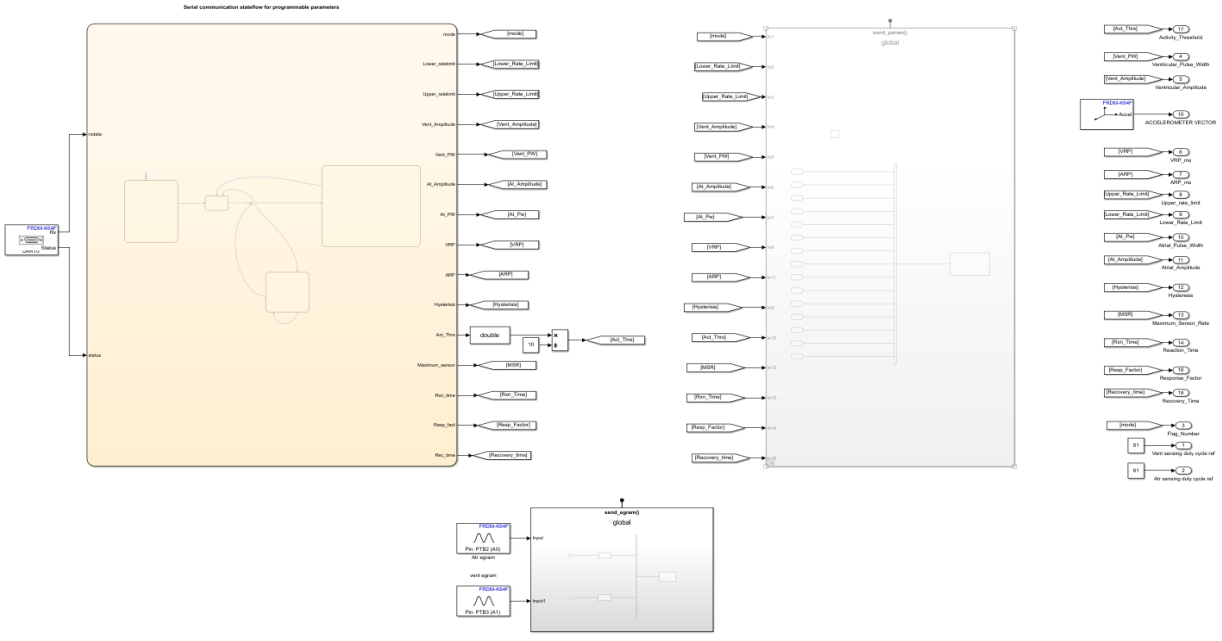

Figure 22: System Inputs Subsystem

Serial Communication

In order to dynamically control the pacemaker shield and the board without restarting the board every time a change is done; the board was connected to the DCM using Serial communication to allow programmable parameters to be sent to the board. Serial communication is one of the features of the NXP-K64F board and can interface with any machine that has a built-in serial port. Serial communication has two sides when communicating the first being from the board and the second being the DCM. In our design there are instances when the board needs to receive information form the DCM, and in other instances, the board is required to send data to the DCM (for the e-gram for example). From the board side, Simulink support package is used to control serial communication which was convenient as the whole logic for the pacemaker is implemented in Simulink and the Simulink support package for K64F has built in serial receive and transmit blocks.

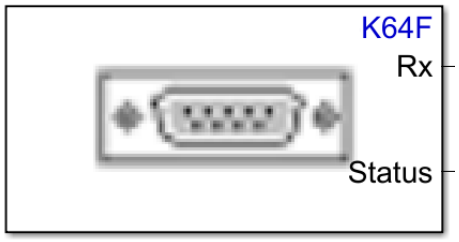

Figure 23: Serial Receive

The Serial Receive block requires specification of the data type that is being sent and the length of the data being sent. In the current implementation the serial receive expects 16 numbers of type uint16. The output of the serial receive block is the array Rx which is an array of the data being sent, and the variable status of type uint8 which is equal to zero when a new packet of data is available. The DCM sends 15 parameters and the first uint16 that is being sent indicates whether the data being sent is the parameters to be used by the pacemaker or whether it is a request from the DCM for the board to send back the values required to display the e-gram. This is done through a flow chart in which if the first value received is 1000 then that indicates that new data is being uploaded to the pacemaker and the data is

assigned to the variables that control the pacemaker logic, however if the first value received is 2000 then that is an indication for the pacemaker to send back the values required for the e-gram until another packet of data is received.

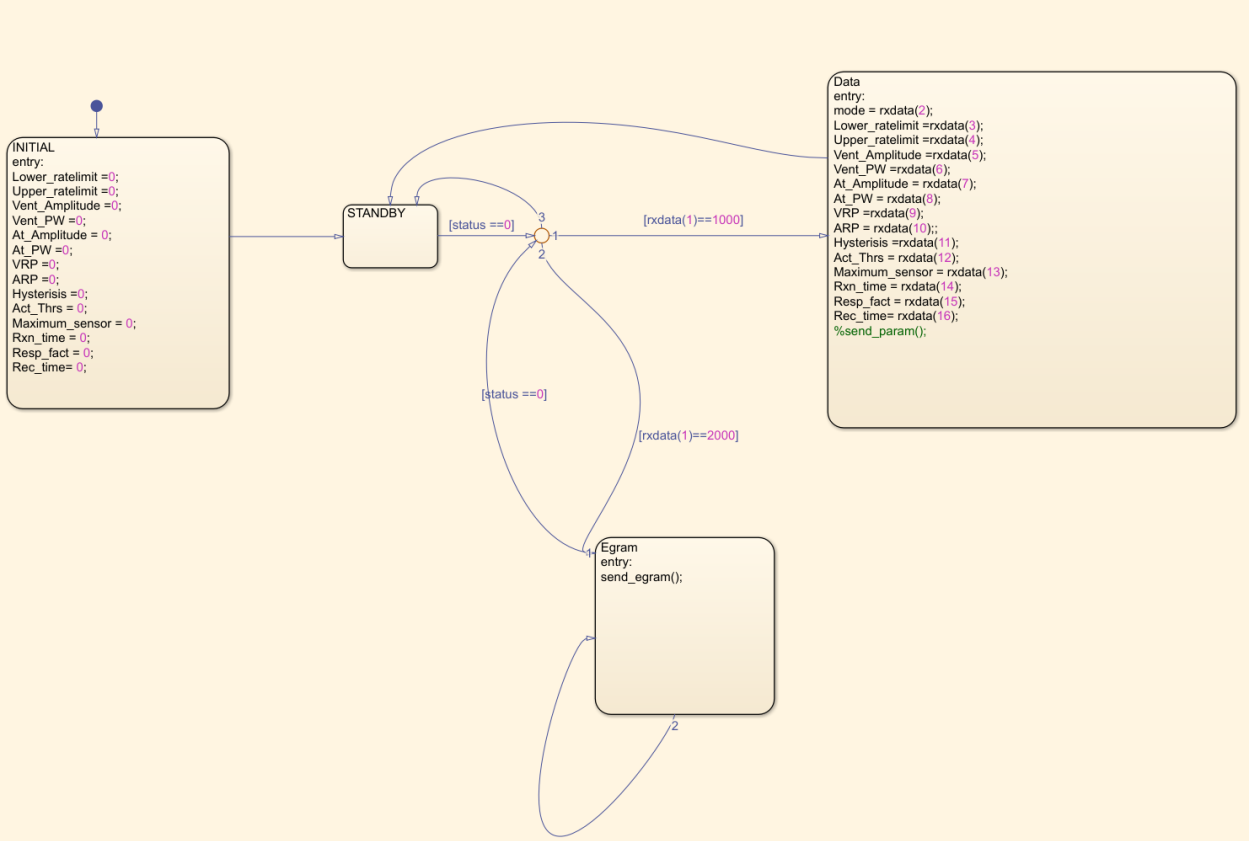

Figure 24: Serial Communication Stateflow Chart

In addition to the Serial receive block, Simulink has a Serial transmit block that is used to send the data back to the DCM. Initially, the serial transmit block was used to send back the parameters received in Simulink in order to confirm the data being sent using the function subsystem send_params()(which is commented out in our Simulink design), however, in order to achieve the e-gram functionality the serial transmit block was used to continuously send the e-gram data through the function subsystem send_egram() which takes the inputs of analog read blocks for pin A0 and A1, which are responsible for reading the data required for Atrial e-gram and Ventricular e-gram respectively.

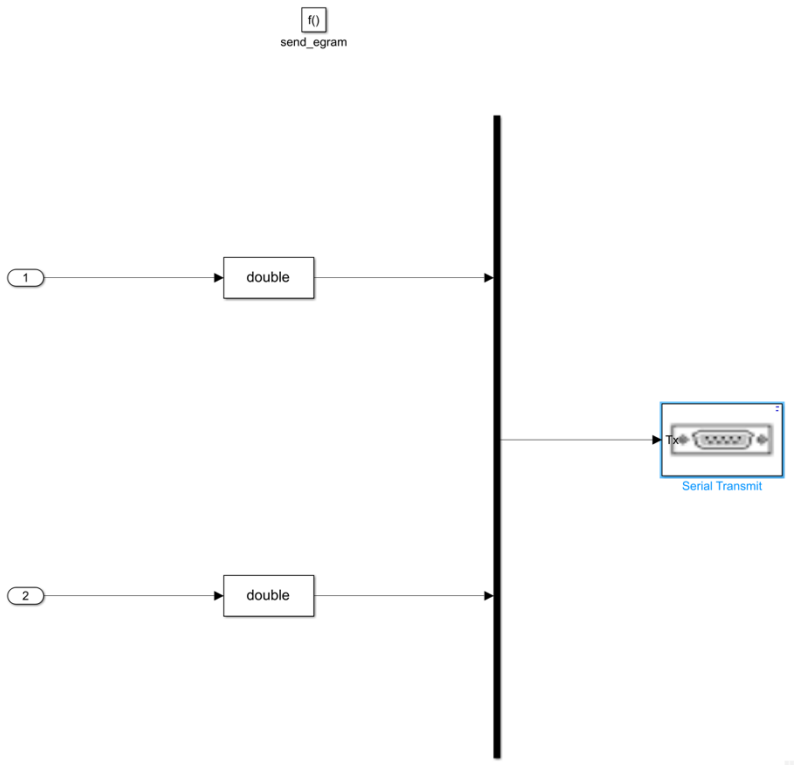

Figure 25: send_egram ()

The send_egram () function expects 2 inputs which converts to type double to ensure the most accuracy and the both inputs are combined in a bus using the mux block which then is fed to the serial transmit block which sends the parameter to the DCM as 2 numbers of type double.

In order to ensure that the data is being transmitted and received properly as serial communication protocol was established to confirm that the DCM and the pacemaker are sending and receiving the appropriate data at the correct timing. When the DCM sends parameters to the pacemaker, it sends 16 uint16 numbers which is 2 bytes each. The data being sent is uint16 as it requires less bytes to transmit the data and is not limited from 0-255 like in uint8. On the other hand, when the pacemaker sends the e-gram data to the DCM it sends 2 numbers of type double in order to ensure maximum accuracy for the decimal places and since we are sending only 2 numbers (each 8 bytes) only 16 bytes are being transmitted from the pacemaker to the DCM. In addition to the data types matching when sending and receiving the data, the order of the data being sent and received is identical on both the DCM and the pacemaker as a result each number transmitted is used properly to achieve a certain functionality. For example, the pacemaker expects the first number send from the DCM to indicate whether a set of new parameters is being sent or a request for the e-gram data is being sent, and then the second number sent is an indicator of the mode that the parameters are being passed to. Similarly, when the e-gram data is sent the DCM expects the first number to be for the Atrial e-gram and that the second number represents the value for the Ventricular e-gram.

This Serial protocol is unique for the data required for assignment 2 as for future aspects of the project this serial protocol might need to be modified as assignment 3 might require a different set of parameters that might be a different type that is not uint16 or double and thus if the current serial

communication protocol is being used then the data transmitted and received will be completely incorrect as the pacemaker would expect a certain number and receive the wrong number and thus perform the incorrect functionality. As a result, depending on the set of parameters that are required for assignment 3 and their corresponding data types the serial communication protocol might need to be modified accordingly in order to send and receive the correct data at the appropriate time.

Appendix

Please note that the following tables and figures were provided to the group in the PACEMAKER and pacemaker_shield_explained documents and are only included in this report as a reference.

A Tables:

Table A.1: Programmable Parameters

Parameter	Programmable Values	Increment	Nominal	Tolerance
Mode	Off DDD VDD DDI DOO AOO AAI VOO VVI AAT VVT DDDR VDDR DDIR DOOR AOOR AAIR VOOR VVIR	—	DDD	—
Lower Rate Limit	30-50 ppm 50-90 ppm 90-175 ppm	5 ppm 1 ppm 5 ppm	60 ppm	±8 ms
Upper Rate Limit	50-175 ppm	5 ppm	120 ppm	±8 ms
Maximum Sensor Rate	50-175 ppm	5 ppm	120 ppm	±4ms
Fixed AV Delay	70-300 ms	10 ms	150 ms	±8 ms
Dynamic AV Delay	Off, On	—	Off	—
Minimum Dynamic AV Delay	30-100 ms	10 ms	50 ms	
Sensed AV Delay Offset	Off, -10 to -100 ms	-10 ms	Off	±1 ms
A or V Pulse Amplitude Regulated	Off, 0.5-3.2V 3.5-7.0 V	0.1V 0.5V	3.5V	±12%
A or V Pulse Amplitude Unregulated	Off, 1.25, 2.5, 3.75, 5.0V	—	3.75V	—
A or V Pulse Width	0.05 ms 0.1-1.9 ms	— 0.1 ms	0.4 ms	0.2 ms
A or V Sensitivity	0.25, 0.5, 0.75 1.0-10 mV	— 0.5 mV	A-0.75 mV V-2.5 mV	±20%
Ventricular Refractory Period	150-500 ms	10 ms	320 ms	±8 ms
Atrial Refractory Period	150-500 ms	10 ms	250 ms	±8 ms
PVARP	150-500 ms	10 ms	250 ms	±8 ms
PVARP Extension	Off, 50-400 ms	50 ms	Off	±8 ms
Hysteresis Rate Limit	Off or same choices as LRL	—	Off	±8 ms
Rate Smoothing	Off, 3, 6, 9, 12, 15, 18, 21, 25%	—	Off	±1%
ATR Mode	On, Off	—	Off	—
ATR Duration	10 cardiac cycles 20-80 cc 100-2000 cc	— 20 cc 100 cc	20 cc	±1 cc
ATR Fallback Time	1-5 min	1 min	1 min	±1 cc
Ventricular Blanking	30-60 ms	10 ms	40 ms	—
Activity Threshold	V-Low, Low, Med-Low, Med, Med-High, High, V-High	—	Med	—
Reaction Time	10-50 sec	10 sec	30 sec	±3 sec
Response Factor	1-16	1	8	—
Recovery Time	2-16 min	1 min	5 min	±30 sec

Table A.2: Pin Layout

Pin Name	Corresponding Name	Functionality
D0	ATR_CMP_DETECT	Used in the sensing circuitry of the atrium. Outputs ON (HIGH) when signal voltage is higher than threshold voltage and OFF (LOW) otherwise (includes 5mV hysteresis).
D1	VENT_CMP_DETECT	Same functionality as in ATR_CMP_DETECT but for the ventricle sensing signal (includes 5mV hysteresis).
D2	PACE_CHARGE_CTRL	Used to start and stop the charging of the primary capacitor (C22). If ON (HIGH) $\rightarrow$ PWM charges C22 If OFF (LOW) $\rightarrow$ PWM disconnected from circuit NEVER allow this signal to be set to HIGH if either ATR_PACE_CTRL and/or VENT_PACE_CTRL are HIGH because then the patient may be directly connected to the PWM signal!
D3	VENT_CMP_REF_PWM	In order to establish a threshold for when the ventricular action potential should be sensed, this pin uses PWM to charge a capacitor that will sustain a constant voltage for comparison. Note that the capacitor voltage is linearly proportional to the Duty Cycle of the PWM input. Use 500 Hz PWM frequency.
D4	Z_ATR_CTRL	This control allows the impedance circuit to be connected to the ring electrode of the atrium. It is used to analyze the impedance of the atrial electrode and the electrical connection between the atrial electrodes and the atrium itself. The output of this circuit is found at the Z_SIGNAL pin. More information regarding lead impedances is mentioned in Section 4 of this document.
D5	PACING_REF_PWM	Used to charge the primary capacitor (C22) of the pacing circuit. The PWM voltage output by this pin saturates to 0-5V and will charge C22 is PACE_CHARGE_CTRL is HIGH. Note that the capacitor voltage is linearly proportional to the Duty Cycle of the PWM input. Use 500 Hz PWM frequency.

D6	ATR_CMP_REF_PWM	Same functionality as in VENT_CMP_REF_PWM but for the atrial action potential. Note that the capacitor voltage is linearly proportional to the Duty Cycle of the PWM input. Use 500 Hz PWM frequency.
D7	Z_VENT_CTRL	This control allows the impedance circuit to be connected to the ring electrode of the ventricle. It is use is identical to Z_ATR_CTRL but for the ventricle. More information regarding lead impedances is mentioned in Section 4 of this document.
D8	ATR_PACE_CTRL	Used to discharge the primary capacitor through the atrium. Current flows through the switch if set to HIGH. If LOW there is no current flow. Pay attention to the direction at which current flows through the electrode. NEVER allow this output signal to be set to HIGH if PACE_CHARGE_CTRL is HIGH because then the patient's atrium may be directly connected to the PWM signal!
D9	VENT_PACE_CTRL	Same functionality as in ATR_PACE_CTRL but for the ventricle. NEVER allow this output signal to be set to HIGH if PACE_CHARGE_CTRL is HIGH because then the patient's ventricle may be directly connected to the PWM signal!
D10	PACE_GND_CTRL	To allow current to flow from the ring to the tip in either the atrium or the ventricle this pin must be HIGH since it controls the switch directly following the tip. Note: Once this pin is activated along with either ...._PACE_CTRL pins, the charge will flow through the switch and accumulate in the blocking capacitor (C21).
D11	ATR_GND_CTRL	Used to connect the ATR_RING_OUT to GND. This functionality is used when discharging the blocking capacitor through the atrium to allow no charge buildup.
D12	VENT_GND_CTRL	Same functionality as in ATR_RING_OUT but for the ventricle.
D13	FRONTEND_CTRL	Used to activate the sensing circuitry. If ON (HIGH) $\rightarrow$ Sensing circuitry will output heart signal. If OFF (LOW) $\rightarrow$ Sensing circuitry is disconnected from patient (Green Connectors) and will output nothing. Note: This switch controls both the atrial and ventrical circuits. It is up to the programmer to only record the data they desire since both will be activated.
GND	GND	References electronic GND. It is good practice to test that all grounds are connected with low resistance prior to the initial startup of the device. Once the device is in use or in the body this process is no longer necessary.

B Figures:

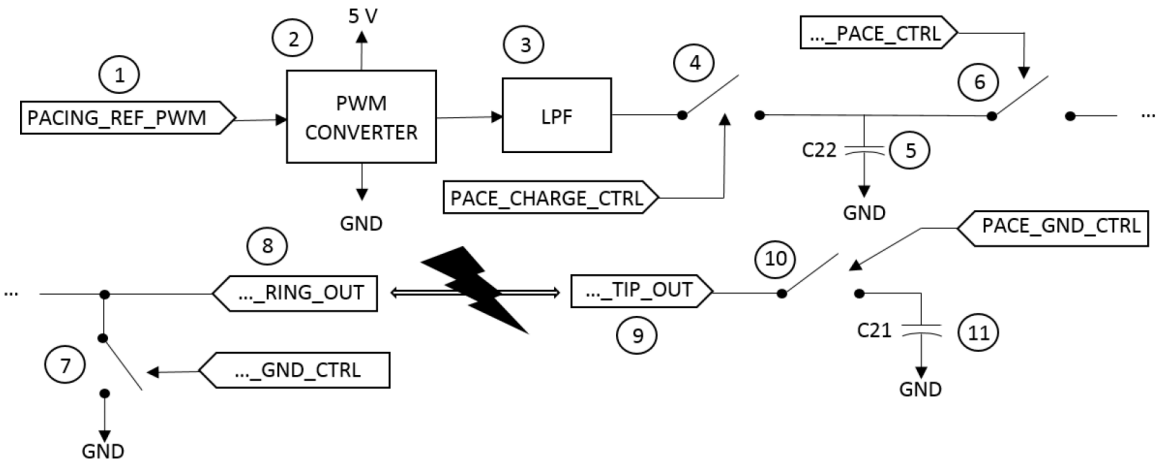

Figure B.1: AV Pacing Circuit Flowchart

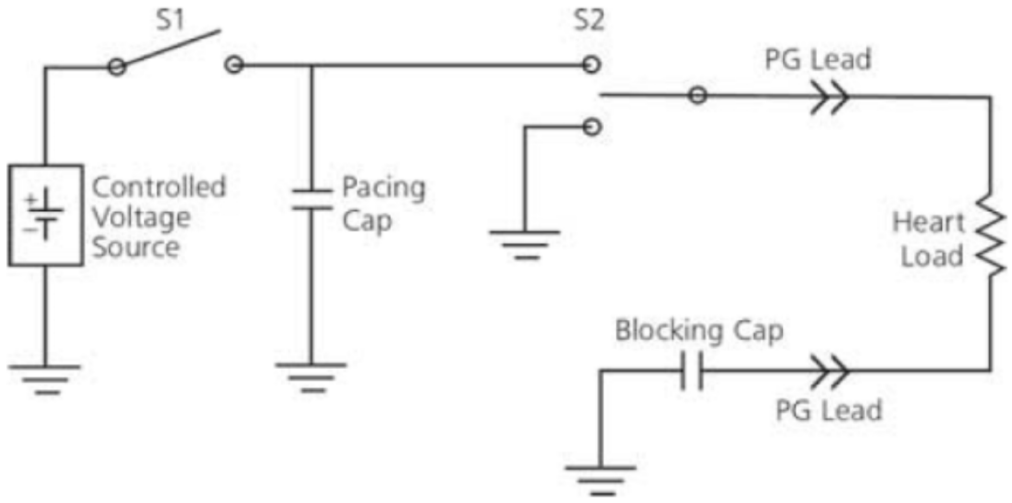

Figure B.2: Simplified Pacing Schematic

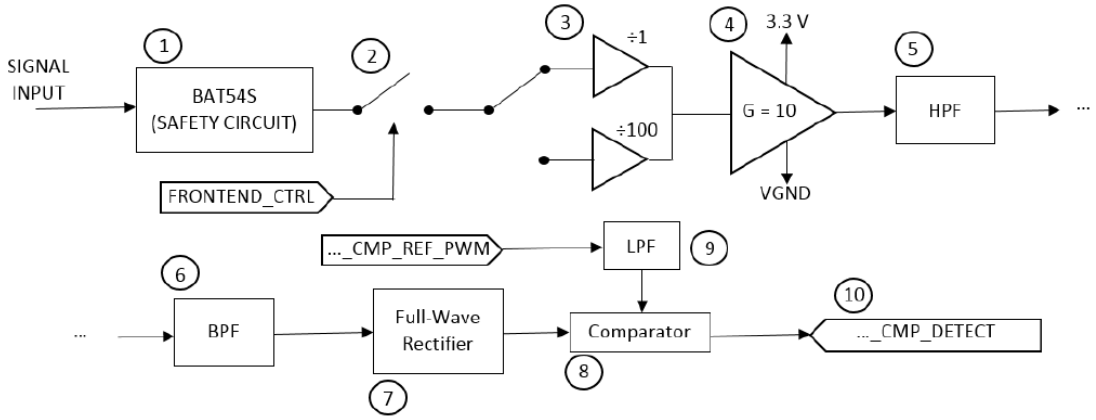

Figure B.3: AV Sensing Circuit Flowchart