SE 3K04

Assignment 2: Part 1

Group 18

Dec 2, 2018

Table of Contents

Input Variables

Convention

p_ ... refers to programmable parameters that can be set by the DCM

r_ ... refers to values that are read through the hardware

l_ ... refers to local variables

Programmable Parameters

Name	Type	Use (Technical)
p_lowrateInterval	uint16	- Input programmable parameter that determines the BPM
p_vPaceWidth	uint16	- Input programmable parameter that determines the pace width for the ventricle
p_aPaceWidth	uint16	- Input programmable parameter that determines the pace width for the atrium
p_vPaceAmp	uint16	- Input programmable parameter that determines the amplitude of the ventricle pace
p_aPaceAmp	uint16	- Input programmable parameter that determines the amplitude of the atrium pace
p_VRP	uint16	- Input programmable parameter that determines the time where a spontaneous pace is not sensed to avoid noise from counting as a pace (for ventricle)
p_ARP	uint16	- Input programmable parameter that determines the time where a spontaneous pace is not sensed to avoid noise from counting as a pace (for atrium)
p_mode	uint8	- Input programmable parameter that determines the pacing mode
r_accel	double	- Magnitude of the input read from the accelerometer on the board
spontPace	boolean	- Evaluates to true when a button is pressed. Used in testing.
VENT_CMP_DETECT	boolean	- See “PACEMAKER Shield Explained” for a detailed explanation.
ATR_CMP_DETECT	boolean	“

Local Variables

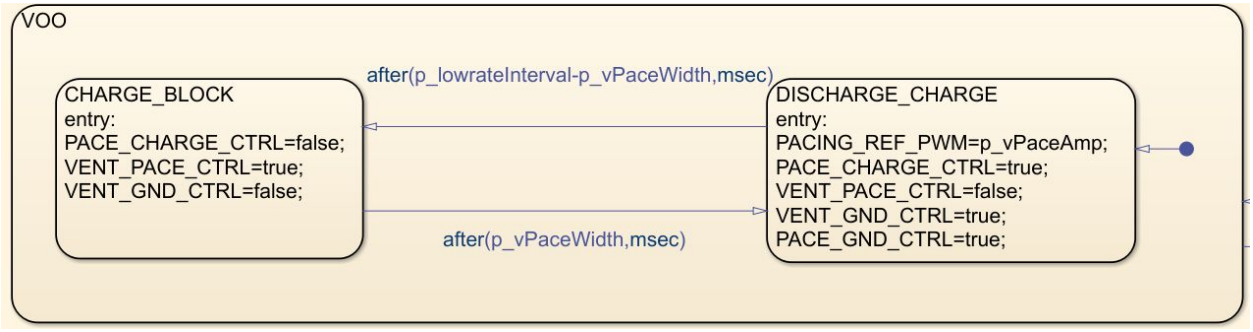
Name	Type	Use (Technical)
l_lowrateInterval	uint16	- Local version of p_lowrateInterval since it is not possible to update an input value
l_setpoint	uint16	- Local variable that is the desired l_lowrateInterval (which converts to BPM)
l_status	boolean	- Local variable counter that is used to determine the status of the user (what activity the user is currently doing)
l_paced	boolean	- Local variable that is true when a pace begins and false otherwise
l_sense	boolean	- Local variable that is true when the system is allowed to sense for a spontaneous pace and false otherwise

Output Variables

Outputs variables are explained in document “PACEMAKER Shield Explained”.

Name	Type	Use (Technical)
PACE_CHARGE_CTRL	boolean	See “PACEMAKER Shield Explained” for a detailed explanation.
VENT_CMP_REF_PWM	double	“
PACING_REF_PWM	double	“
ATR_CMP_REF_PWM	double	“
ATR_PACE_CTRL	boolean	“
VENT_PACE_CTRL	boolean	“
PACE_GND_CTRL	boolean	“
ATR_GND_CTRL	boolean	“
VENT_GND_CTRL	boolean	“
FRONTEND_CTRL	boolean	“
GREEN_LED	boolean	Controls the state of the Green LED in an RGB LED
RED_LED	boolean	Controls the state of the Red LED in an RGB LED
BLUE_LED	boolean	Controls the state of the Blue LED in an RGB LED

Simple Pacing Modes Explained (VOO, AOO)



VOO Implementation

VOO was implemented in our design through the use of two states - a CHARGE_BLOCK state and a DISCHARGE_BLOCK state. In the CHARGE_BLOCK state, capacitor C21 is being charged by opening switches PACE_CHARGE_CONTROL and VENT_GND_CTRL, hence them being set to false. VENT_PACE_CTRL and PACE_GND_CTRL are set to true and both switches are closed to allow current to flow across the heart. The time it takes for C21 to charge is determined by the parameter p_vPaceWidth which is a parameter that controls how long the pace should be. In the DISCHARGE_BLOCK, capacitor C21 is discharged and C22 is charged though the closing of switches VENT_GND_CTRL and PACE_CHARGE_CTRL. The period of time in which the design waits to send charge back across the heart from C22 to C21 i s controlled by the desired BPM. The desired BPM is calculated through p_lowrateInterval - p_vPaceWidth.

AOO Implementation

The design for AOO was identical to VOO, only the outputs modified and the programmable parameters used were different. Instead of p_v..., p_a... parameters were used; and instead of VENT_... pins, ATR_... pin values were modified as the output.

(Detailed explanation of VOO/AOO Implementation is documented in assignment 1)

Rate-Adaptive Pacing Explained (VOOR, AOOR)

Introduction

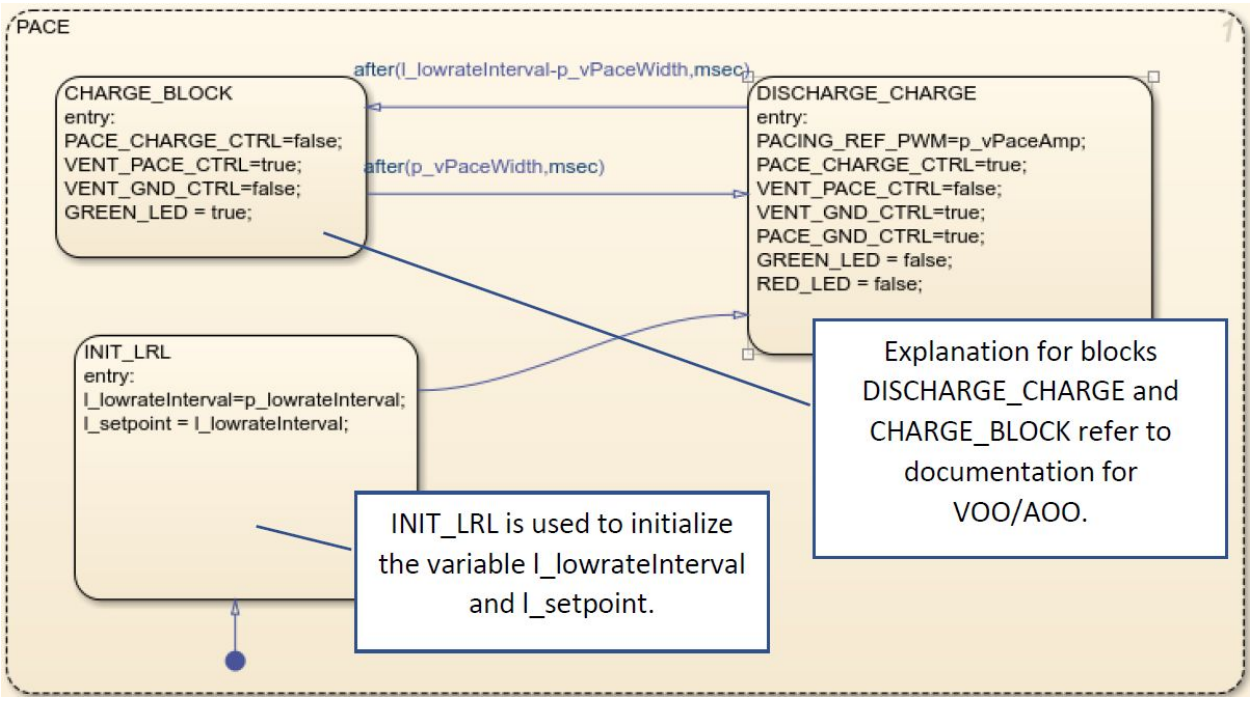
To implement adaptive pacing, first we looked at what is required for the pacemaker to be able to change pacing at a steady rate and how it is related to the accelerometer. It was noted that there were 3 key elements that were needed to have proper pace adapting:

1. Steady increase/decrease in the BPM
2. A target BPM
3. A way to determine the target BPM

The requirements listed above were achieved by implementing 3 subsystems. Each met one of the set requirements.

PACE

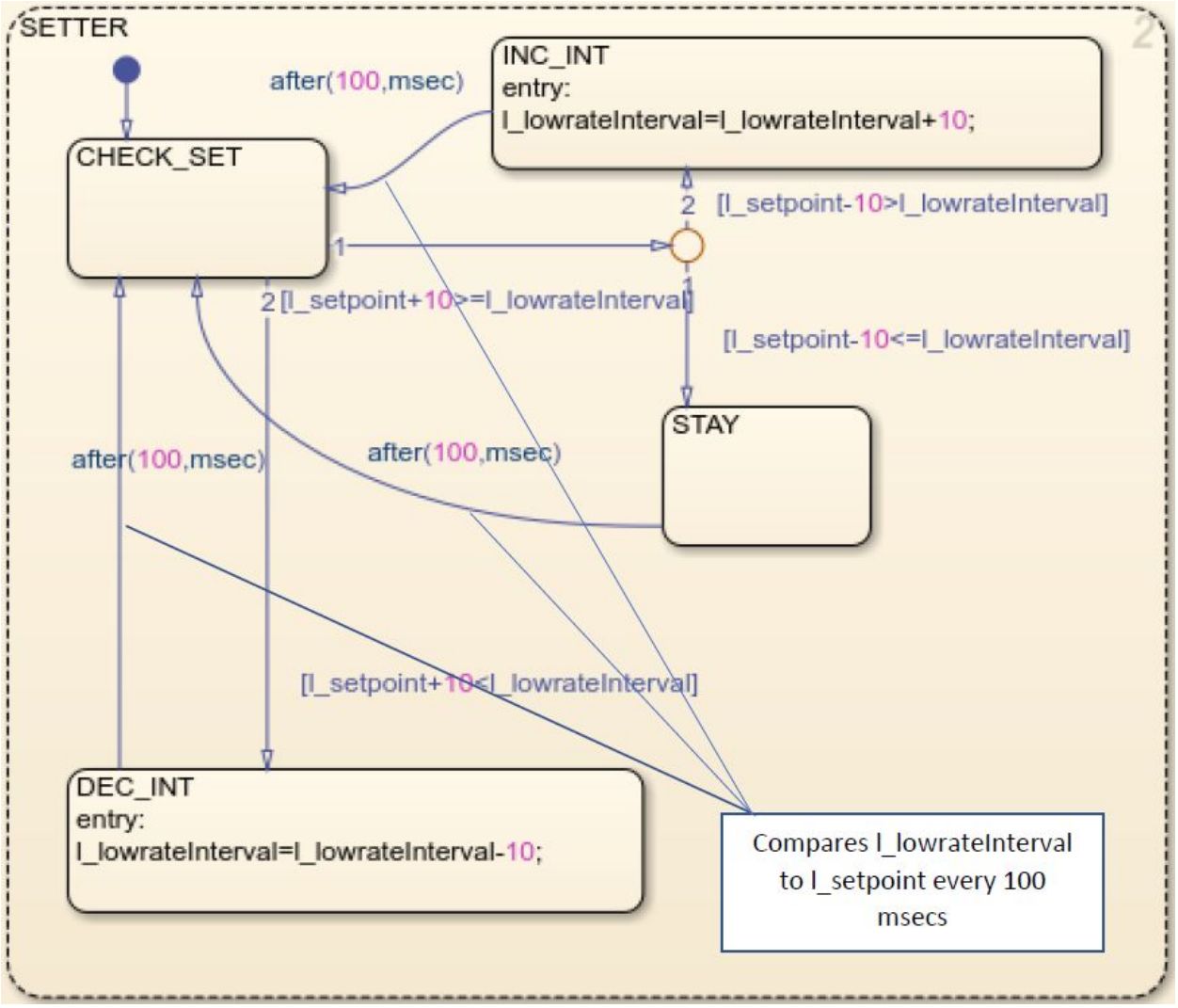
The PACE block paces the specified part of the heart. In the case below it paces the ventricle. It uses parameters l_lowrateInterval, p_vPaceWidth and p_vPaceAmp.



INIT_LRL is needed because without initializing the local variables the program crashes as it is trying to compare and modify the variables in other parts of the code.

SETTER

The SETTER block adjusts l_lowrateInterval to approach l_setpoint or maintains the value of l_lowrateInterval if it is within a specified range of l_setpoint.



This block is important because it ensure that the BPM changes gradually and does not jump (which would be dangerous for the user).

Overview of how the conditions work:

If l_lowrateInterval is within or equal to +/- 10 msec of l_setpoint, the value of l_lowrateInterval is maintained (STAY block).

If l_lowrateInterval is lower than l_setpoint - 10 msec then l_lowrateInterval is increased by 10 msec (INC_INT block)

If l_lowrateInterval is greater than l_setpoint + 10 msec then l_lowrateInterval is decreased by 10 msec (DEC_INT block)

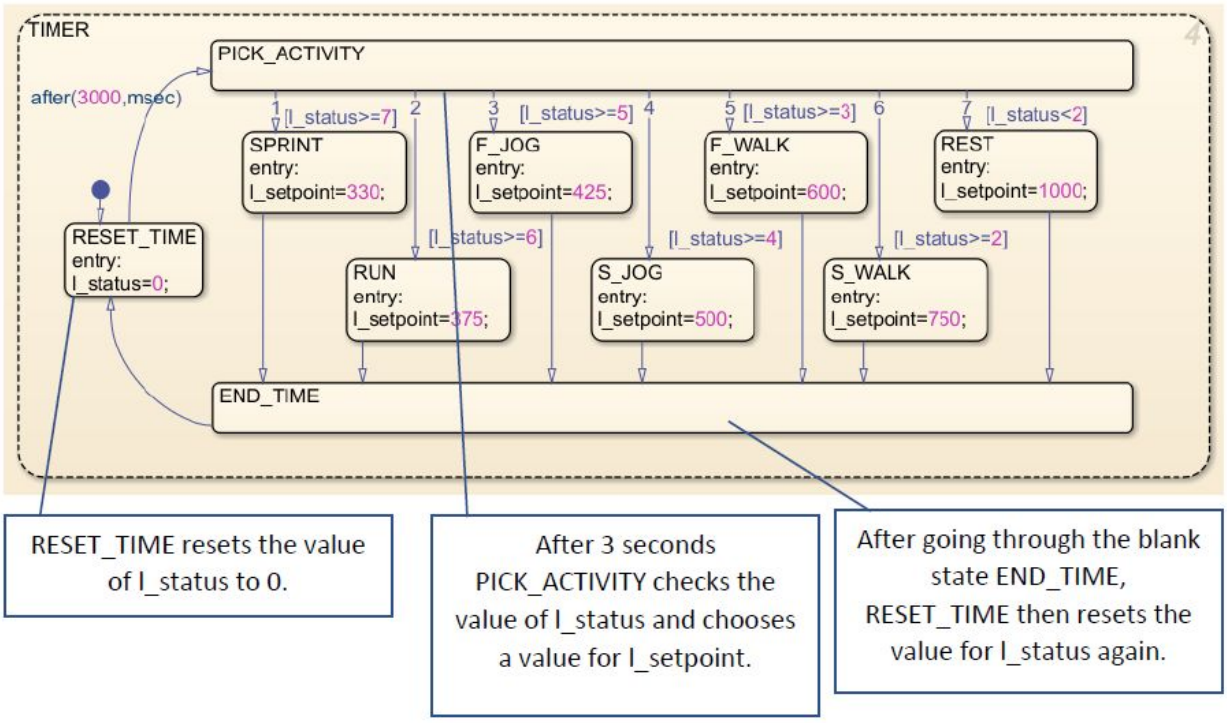
Function Table

		l_lowrateInterval (new)
l_setpoint + 10 >= l_lowrateInterval	l_setpoint-10<=l_lowrateInterval	Do nothing
	l_setpoint-10>l_lowrateInterval	l_lowrateInterval + 10
l_setpoint +10 < l_lowrateInterval		l_lowrateInterval - 10

* Incrementing occurs every 100 msec so that the l_lowrateInterval doesn't jump to the value of l_setpoint in a very small time frame. Checking at 100 msec allows for a steady increase/decrease in the l_lowrateInterval (which in turn causes steady change in BPM)

TIMER

The TIMER block chooses a value for l_setpoint every 3 seconds.



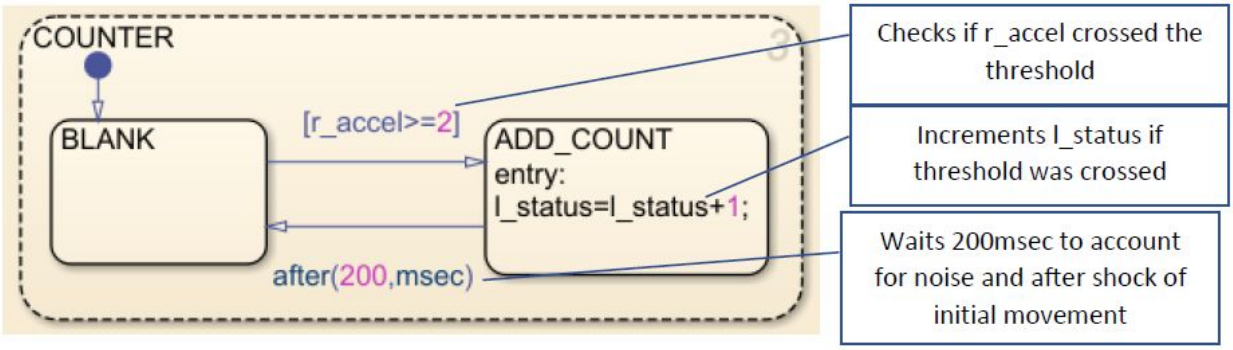
The process for the TIMER block records how much movement has occurred in a given time interval and classifies it as an activity that the user is likely doing. Based on the activity it sets a target for the BPM. Without this block there would be no cap on how much the BPM changes by and it would be difficult to determine if the BPM should go up or down.

Function Table

	l_setpoint
l_status >= 7	330
l_status >= 6	375
l_status >= 5	425
l_status >= 4	500
l_status >= 3	600
l_status >= 2	750
l_status <2	1000

COUNTER

COUNTER checks if movement has occurred. The block acts like a step counter. Each time a step is taken there is impulse that occurs on the feet which creates a spike in the acceleration. The block counts the number of steps by incrementing an integer every time the accelerometer crosses a threshold (each time it spikes).



Each time r_accel crosses the threshold, l_status is incremented by 1. There is a 200 msec period where it does not look for movement to avoid including continuous movement from the accelerometer after the initial movement. The COUNTER block works with the TIMER block to get a frequency of the movement by the user.

Function Table

	l_status
r_accel >=2	l_status + 1
r_accel <2	Do nothing

AOOR

The design for AOOR was identical to VOOR, only the outputs modified and the programmable parameters used were different. Instead of p_v..., p_a... parameters were used; and instead of VENT_... pins, ATR_... pin values were modified as the output.

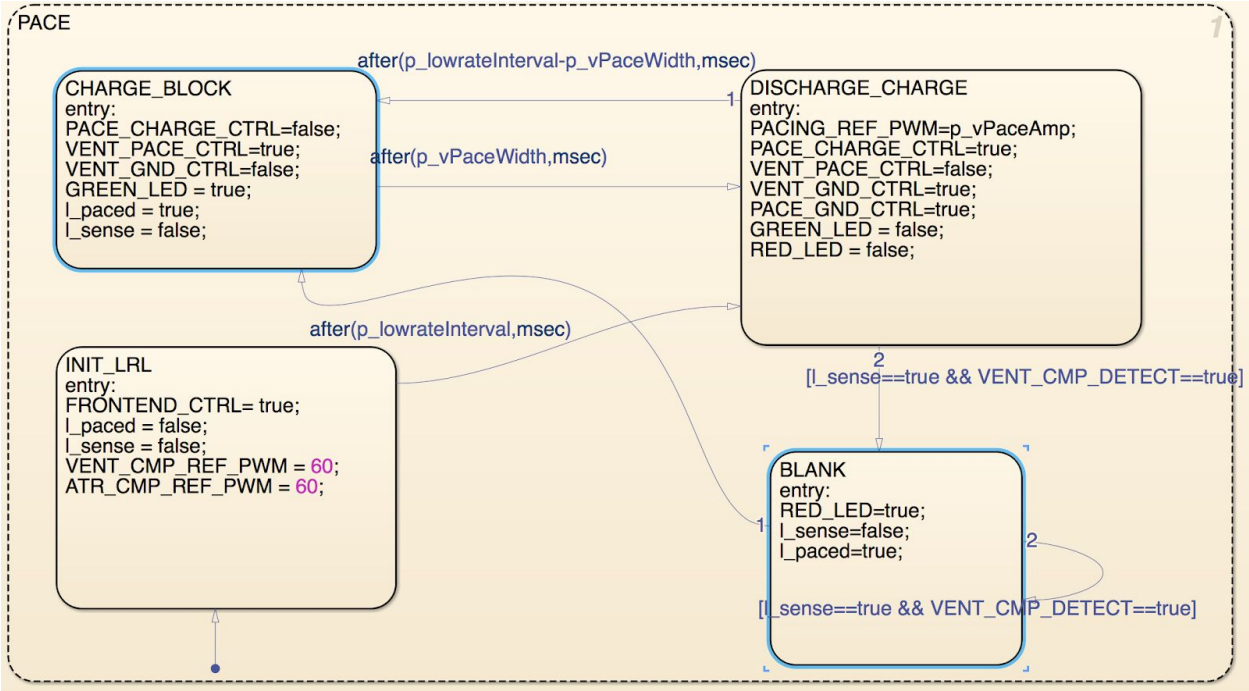
Pace Inhibition Explained (VVI, AAI)

Introduction

In order to implement pace inhibition, the pacemaker is required to stop pacing the heart when it senses a natural heartbeat, and pace the heart if the natural heartbeat is weak or nonexistent. In addition, a refractory period after any pacing must be implemented so that the heart does not receive two paces in quick succession, which could damage the heart or lead to arrhythmia or tachycardia.

The PACE and REFRACTORY subsystems implement inhibition and a refractory period, respectively.

PACE



Function Table

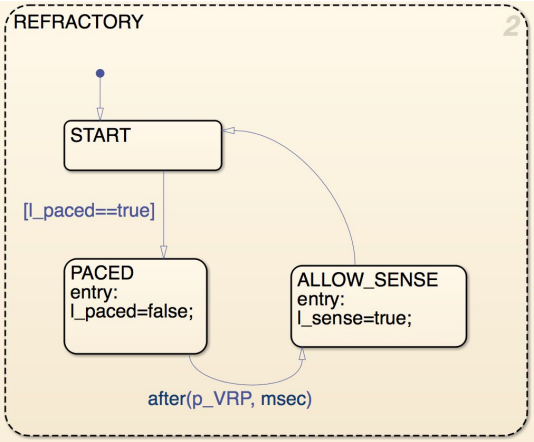
		I_paced
I_sense==true	VENT_CMP_DETECT==true	true
	VENT_CMP_DETECT==false	false
I_sense==false		false

The implementation shown in the diagram above is similar to VOO with some modifications. For more details on the CHARGE_BLOCK and DISCHARGE_CHARGE states, refer to “Simple Pacing Modes Explained”.

The first state, INIT_LRL, sets the FRONTEND_CTRL to HIGH, since the FRONTEND_CTRL is set to false every time p_mode is changed. ... _CMP_REF_PWM is set to 60 after experimenting with the sensing circuit. It is concluded that to sense a natural beat the reference voltage must be at 60%. This is added into the state because this value is not likely to change. For the purpose of this project, we assume that the fluctuation of the intensity of the beats is not significant.

The addition of the BLANK state is what provides the inhibition functionality. From the DISCHARGE_CHARGE state, if the input variable VENT_CMP_DETECT is true (and the refractory period is over), the system ceases any pacing and turns on the RED_LED to indicate pace inhibition. The state loops to itself if another natural pace is detected, or paces the heart if no beat is detected within the p_lowrateInterval.

REFRACTORY



The REFRACTORY state and its local variables, l_paced and l_sense, prevent the heart from being paced again during the refractory period after either a natural or an artificial pace has just occured. This state runs in parallel with PACE and waits for the l_paced variable to be set to true, indicating that the heart was paced (shown in the blue-outlined states of the PACE subsystem diagram). l_paced is then set back to false to reset it.

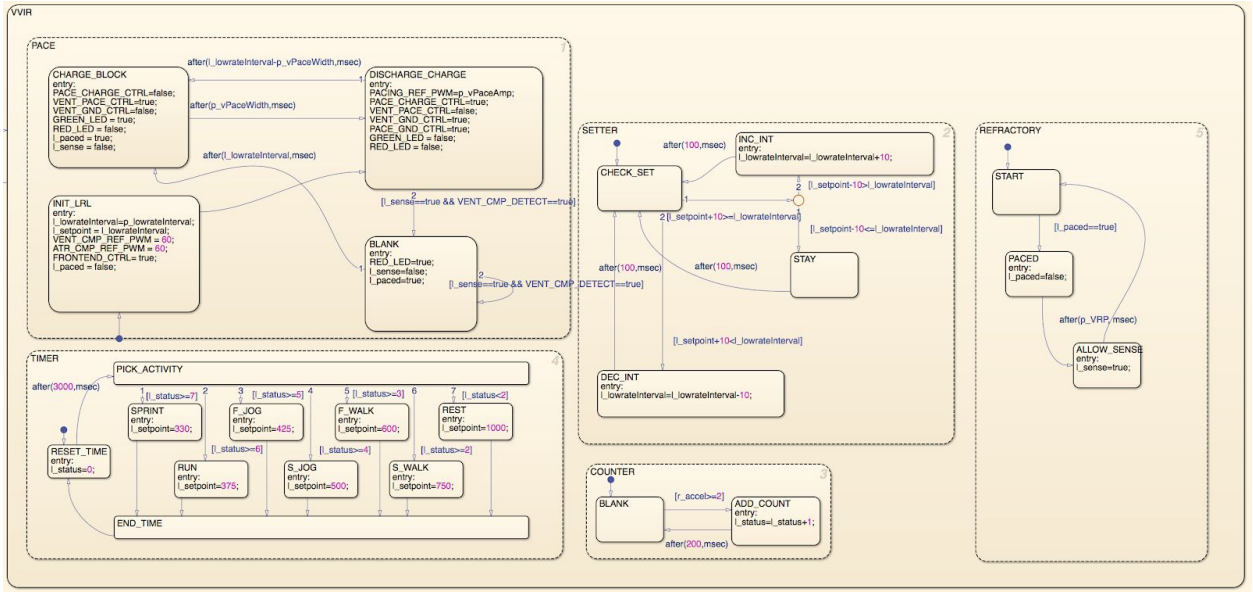
After a pace, the system waits for p_...RP before setting l_sense to true, allowing the system to accept new sensing from ..._CMP_DETECT. This prevents any artificial pacing during natural paces outside of the refractory period, while the heart is running normally. In the BLANK and CHARGE_BLOCK states, l_sense is then set back to false to reset it. It is reasonably assumed that p_...RP is less than l_lowrateInterval, which ensures that an artificial pace will only happen after the refractory period has elapsed.

AAI

The design for AAI was identical to VVI, only the outputs modified and the programmable parameters used were different. Instead of p_v..., p_a... parameters were used; and instead of VENT_... pins, ATR_... pin values were modified as the output.

Rate-Adaptive and Inhibited Pacing Modes Explained (VVIR, AAIR)

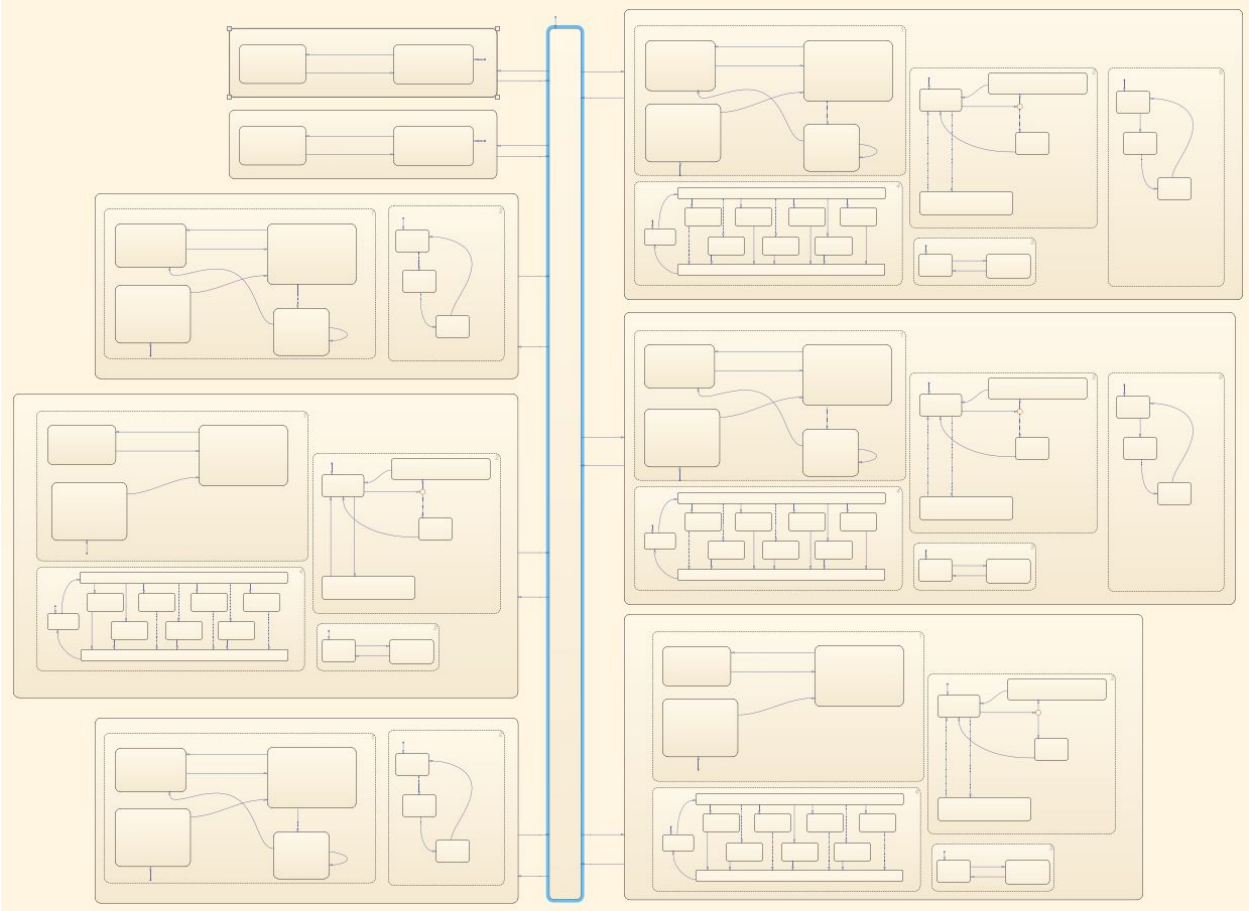
The rate adaptive functionality, implemented in VOOR and AOOR, and inhibition functionality, implemented in VVI and AAI, are designed to work without interfering with the operation of each other. Thus, they can both be run concurrently to achieve VVIR and AAIR. Refer to the above two mode types for further details on how rate-adaptation and inhibition are implemented.



Layout

The layout of the simulink stateflow is shown below. Each mode required is encapsulated within its own state with a top-level state being able to pass control to the proper mode. This top-level state can be seen in the below diagram as the state highlighted in blue. Each of the large states that sprout off from this top-level state were given a number to represent what mode they would be. This number corresponds to the number the DCM sends to the program when a mode is selected. This format was required as when one mode is selected, it needs to have sole control of the operation of the program. In the diagram it can be seen that each mode's state also has a return state transition arrow going back to the top-level state. This makes it possible for the DCM to switch the pacemakers board without having to completely reprogram it which was a requirement given in the assignment specifications.

This design layout makes it possible to work on one mode specifically and it is easier to troubleshoot if something does not work. It is also easily readable and easy to walkthrough. Additionally the modes work independently so creating new modes is easier since you can use copies of existing blocks and add to them. Weaknesses include that if a something that is common for all modes is changed, then the update must manually done in all other places. This weakness can be remedied by implementing constants as input variables to the chart, as explained in the design changes below.



Future Requirements and Design Changes

There are a few changes to the requirements for the next assignment. Most of the functionality has already been implemented, however three different functionalities will have to be added.

Our design will need to have a working implementation of DDD and DDDR modes. To do this, an AV delay, which is the natural delay between two chambers, will have to be integrated into the design.

In the current design there are multiple states that change l_setpoint to various numbers for different activities such as walking or running. A better design would have these values as constant input that can be modified for individual users, instead of having it hardcoded. This would make changes to the constants easier to manage.

In addition, hardware hiding to the PACEMAKER chart is not fully complete, as the two inputs used for sensing (..._CMP_DETECT) use blocks directly from the K64F, instead of an input hardware hiding subsystem. The hardware hiding of all output variables, on the other hand, is complete.

References

- “3K04 Requirements Specification of the Pacemaker”, “srsVVI” Wassyng, 2008
- “Functionality and Circuitry Explanation: Pacemaker Microcontroller Shield”, “Pacemaker Shield Explained” Meyer, 2018
- “PACEMAKER System Specification” Boston Scientific, 2007
- “Simulink Serial COM Guide” Ayesh, 2017
- “Advanced Timing Cycles” Wassyng, 2018