Assignment 2 Specific Documentation

Document how programmable parameters originate at the DCM and are implemented in the device.

The values come from the DCM through a form that the user fills out. The user first clicks the “Save” button which ensures all values are allowed and saves the form to a local database. Only once the “Save” button has been clicked and approved, does the “Apply” button become enabled. (It is greyed out before). When the user clicks “Apply”, the form values then get packaged and sent to the FRDM board over the serial port.

From the simulink side the data comes in through the Rx block and then gets interpreted in the simulink stateflow within the inputs subsystem. After the data is indexed and put into their proper parameters, the values then get output to the rest of the simulink model.

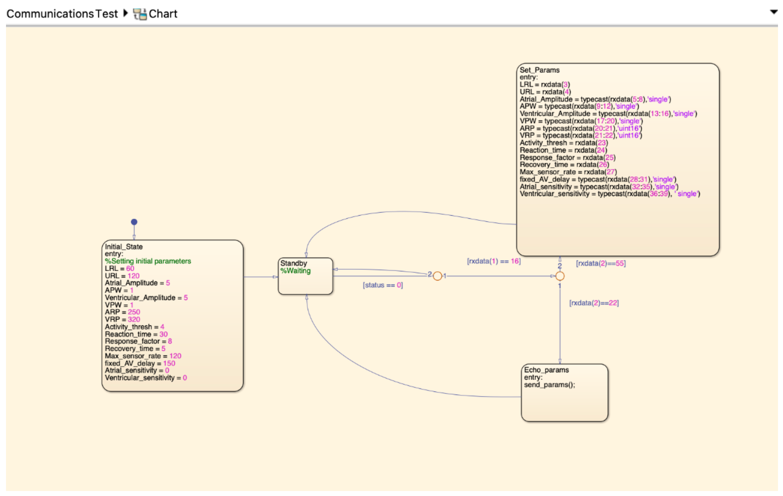
When the echo parameter is called, it calls the send_params() global function and all of the parameters (with the Egrams data) gets routed through the multiplexer and sent over serial through the tx block in the simulink block.

The DCM can then read the incoming message through the “echo” and index the byte stream at the proper indices to form a dictionary of programmable parameters that were stored on the board.

After the dictionary is constructed it can be compared to the variables in the DCM.

Show how you can ensure the parameters stored in the Pacemaker are what the doctor input on the DCM.

An important feature is to ensure that what the doctor has inputted is the same as what is actually being implemented on the Pacemaker as this can have severe consequences. To allow for this, we have implemented an ‘echo’ function in our serial communication. On the Simulink side, to looks as follows:



```
Initial_State
entry:
%Setting initial parameters
LRL = 60
URL = 120
Atrial_Amplitude = 5
APW = 1
Ventricular_Amplitude = 5
VPW = 1
ARP = 250
VRP = 320
Activity_thresh = 4
Reaction_time = 30
Response_factor = 8
Recovery_time = 5
Max_sensor_rate = 120
fixed_AV_delay = 150
Atrial_sensitivity = 0
Ventricular_sensitivity = 0
```

```
Standby
%Waiting
```

```
Set_Params
entry:
LRL = rxdata(3)
URL = rxdata(4)
Atrial_Amplitude = typecast(rxdata(5:8),'single')
APW = typecast(rxdata(9:12),'single')
Ventricular_Amplitude = typecast(rxdata(13:16),'single')
VPW = typecast(rxdata(17:20),'single')
ARP = typecast(rxdata(20:21),'uint16')
VRP = typecast(rxdata(21:22),'uint16')
Activity_thresh = rxdata(23)
Reaction_time = rxdata(24)
Response_factor = rxdata(25)
Recovery_time = rxdata(26)
Max_sensor_rate = rxdata(27)
fixed_AV_delay = typecast(rxdata(28:31),'single')
Atrial_sensitivity = typecast(rxdata(32:35),'single')
Ventricular_sensitivity = typecast(rxdata(36:39), ' single')
```

```
Echo_params
entry:
send_params();
```

This image is of our stateflow for serial communication. As can be seen here, there is an initial state, a standby state, a set parameter state and an echo parameter state. On this, the set parameter state takes in the rx_data input from the dcm serial communication which gives us values to save to each of these variables. Once these values were set, the echo parameter state here sends back these values to the user which allows them to read them. This sends these values back through serial communication to the DCM.

The values get sent back via the UART serial communication from the Simulink to the DCM. This sets the values inside a dictionary. Next, the values in the dictionary which have been transported from the Simulink to the DCM get compared to the values in the DCM to ensure that all of these are the exact same. If they're not the same, an error will be raised and tell the doctor to reset the board and check his values.

Justify your choice of the data types used to represent parameter data.

Parameter	**Minimum value**	**Maximum value**	**Increments**	**Unit**	**Data type**	**Number of bytes**	**Justification of data type**
LRL	30	175	5,1,5	ppm	uint8	1	LRL is a positive integer value, ranging from 30 to 175 ppm. It has a maximum value of less than 255. Therefore, it can fit within uint8 which is only 1 byte.
Atrial amplitude	100	5000	100	mV	uint16	2	Atrial amplitude is a positive float value, ranging from 0 to 5 V with an increment of 0.1. Since floats take extra bytes (4 bytes), we decided to convert atrial amplitude to mV so that it would always be an integer value. Atrial amplitude in mV has a maximum value of 5000 mV which is greater than the range of a uint8 (0 to 255) but less than that of a uint16 (0 to 65535). Therefore, we can represent atrial amplitude with a uint16, which is 2 bytes. This saves 2 bytes compared to if we used floats.
APW (atrial	1	30	1	ms	uint8	1	Same rationale as LRL

pulse width)							
Ventricular amplitude	0	5000	100	mV	uint16	2	Same rationale as atrial amplitude
VPW (ventricular pulse width)	1	30	1	ms	uint8	1	Same rationale as LRL
ARP	150	500	10	ms	uint16	2	ARP is a positive integer value, ranging from 150 to 500. It has a maximum value greater than 255, so we could not use uint8. Therefore, we used uint16. One could argue that we could have converted to centiseconds instead of milliseconds, so that we could use uint8's instead of uint16's; however, since this only saves 1 byte, we decided it would not be worth the trade-off.
VRP	150	500	10	ms	uint16	2	Same rationale as ARP
ASens	0	5000	100	mV	uint16	2	Same rationale as atrial amplitude
VSens	0	5000	100	mV	uint16	2	Same rationale as atrial amplitude
Activity Threshold	1	7	N/A	N/A	uint8	1	Instead of saving Activity Threshold as a string, which would take a lot of bytes, we decided to convert Activity Threshold to a value between 1 and 7, inclusive. Following the rationale of LRL, we decided to use uint8 for activity threshold, as this would take up the least number of bytes.
Reaction Time	10	50	10	ms	uint8	1	Same rationale as LRL
Response Factor	1	16	1	N/A	uint8	1	Same rationale as LRL
Recovery Time	2	16	1	min	uint8	1	Same rationale as LRL
Maximum Sensor Rate	50	175	5	ppm	uint8	1	Same rationale as LRL
Fixed AV Delay	70	300	10	ms	uint16	2	Same rationale as ARP
PVARP	150	500	10	ms	uint16	2	Same rationale as ARP
Mode	1	10	1	N/A	uint8	1	Instead of saving the mode (eg. "AOO", "VOO", etc.) as a string, which would take a lot of bytes, we decided to convert the mode to a value between 1 and 10, inclusive. Following the rationale of LRL and activity threshold, we decided to use uint8 for mode, as this would take up the least number of bytes.

Additional DCM Documentation

Requirements

- The latest version of Python
- Pillow
    - Install with `pip install pillow`
- pySerial
    - Install with `pip install pySerial`
- Matplotlib
    - Install with `pip install matplotlib`

Instructions for launching DCM

1. Clone the repository
2. Open terminal and navigate to the DCM_group44 folder.
   `cd DCM_group44`
3. Launch the DCM by running main.py through the terminal.
   `python3 main.py`

Explanation of each file and class organization system

`Main.py (window):` the start file or "welcome page" that can call the login window or registration window upon the respective button being clicked. Defines and calls a Refresh function upon button click to check for connection with pacemaker board and update the values in SerialCommunication accordingly (will be explained below).

`Login.py (frame):` launched from the main file. Checks if inputted credentials are valid and exist in the database, and launches modeSelection, passing on the username so that the modeSelection window knows what row in the database to import values from and save values to.

`Registration.py (frame):` launched from the main file. Checks if inputted credentials are valid and saves them to the database. It launches modeSelection, passing on the username for the same reasons as Login.py above.

`Patient.py:` defines the Patient class whose instance stores the values and parameters for a patient, as well as methods to copy values from the database and save values to the database.

`data.py:` defines the createDB() function which is called by several window scrips to initialize a database file if it doesn't already exist.

`modeSelection.py (frame):` launched from either registration.py or login.py. Instantiates a Patient object with default parameter values and calls the method

Patient.copyFromDB() which overwrites those default values with any custom values in the database that relate to the patient with that username. The window shows 10 modes to choose from: AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, VVIR, DDD, DDDR. When one of the modes is chosen and “Next” button is clicked, it launches a specific frame from the 4 frames in pacingModes.py. Transitions between modeSelection.py and pacingModes.py are switching frames instead of switching windows. When launching pacingModes, the Patient object stored in this class is transferred to the new pacing mode frame launched, giving pacingModes.py the info needed about the patient in order to copy from and to save to the database.

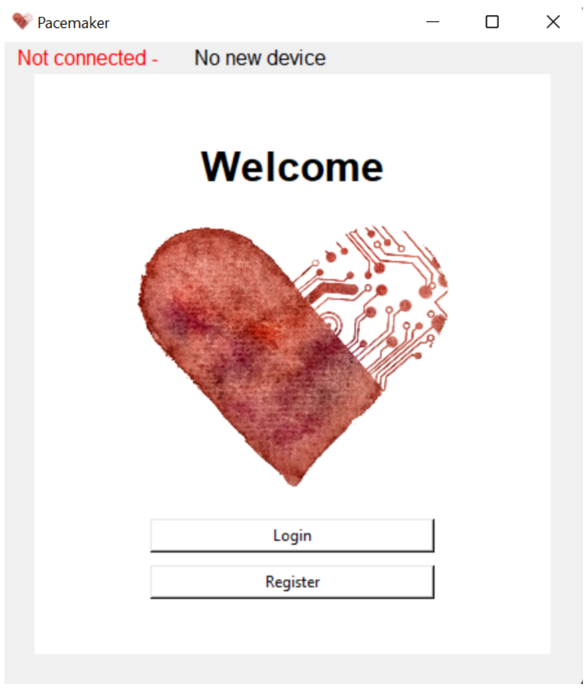
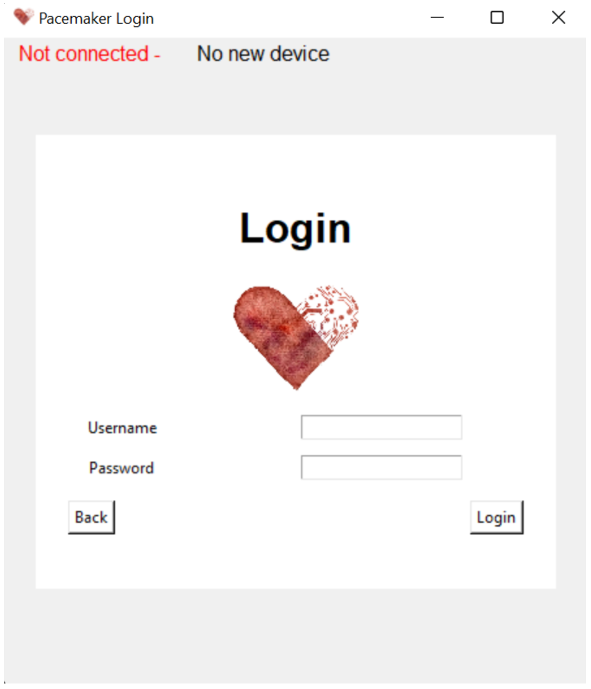
`pacingModes.py (frames):` launched from modeSelection.py from which it gets a Patient object with the values relating to a specific patient in the database. Ten mode frames exist: AOO, VOO, AAI, VVI, AOOR, VOOR, AAIR, VVIR, DDD, DDDR, which are all children of a parent class PacingMode. The parent class PacingMode defines methods to add each label and entry, and the children objects determine which of those methods to call depending on which parameter entries are relevant to that mode. Each frame displays the specific fields, labels, and checkboxes required for the mode it represents. Upon launching, it copies all the values from the Patient object received from the modeSelection window to the entries in that frame. In other words, when a user logs in, they would find their own values in the entries instead of the default values. When the “Confirm” button is clicked, all entry values are transferred to a Patient object, and Patient.saveToDB() method is called, transferring the patient information to the database.

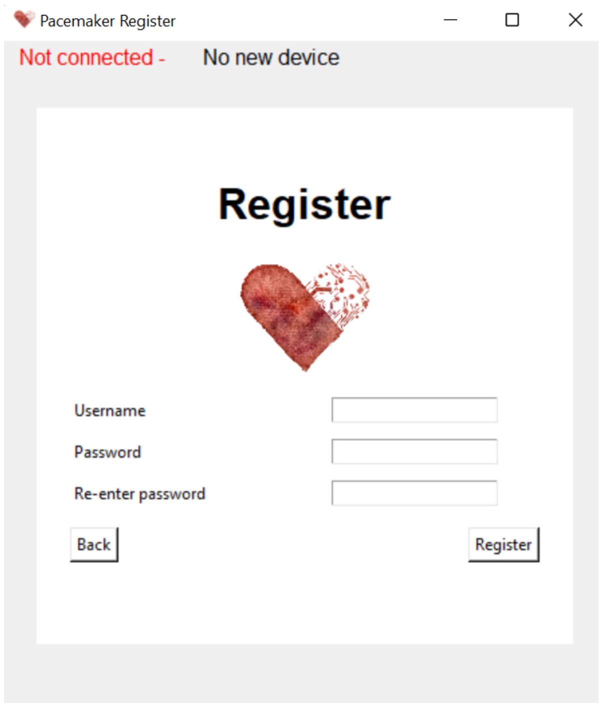
`connectionDisplay.py:` defines two functions, displayConnection and displayNewDevice, which display the two global functions defined in that file, connectionChecker and newDeviceChecker. These two variables can be obtained or changed from any other file. It’s important to note that the device is considered “new” when it has never been interacted with before. In other words, the “new device connected” label will still be displayed for the whole duration of the first interaction with that specific pacemaker.

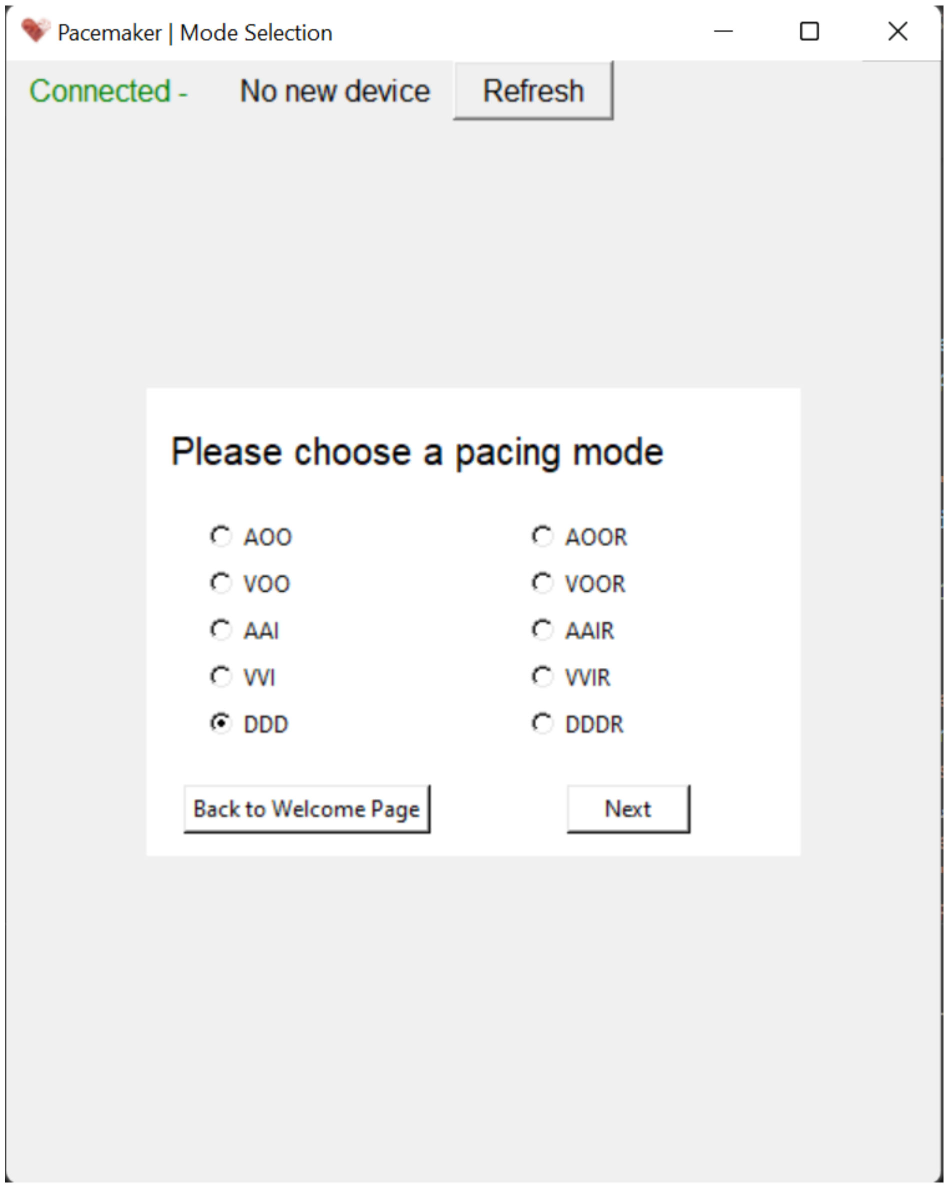
`SerialCommunications.py:` defines a function getPortName that uses the PID and VID of pacemaker boards to find the port name to which the pacemaker board is connected to. It automatically updates the connectionChecker variable in connectionDisplay.py when necessary. This file also defines SerialObject class, which creates a Serial object (serial connection) using getPortName, and opens it. It also includes the SendData method, which takes a patient object and sends the patient parameters to the pacemaker, the PackData method which translates pacing mode into an integer recognized by the Simulink program, and other methods that are associated with serial communication and egrams. The code uses time.sleep(1) to give the program time to switch between writing and reading to avoid issues.

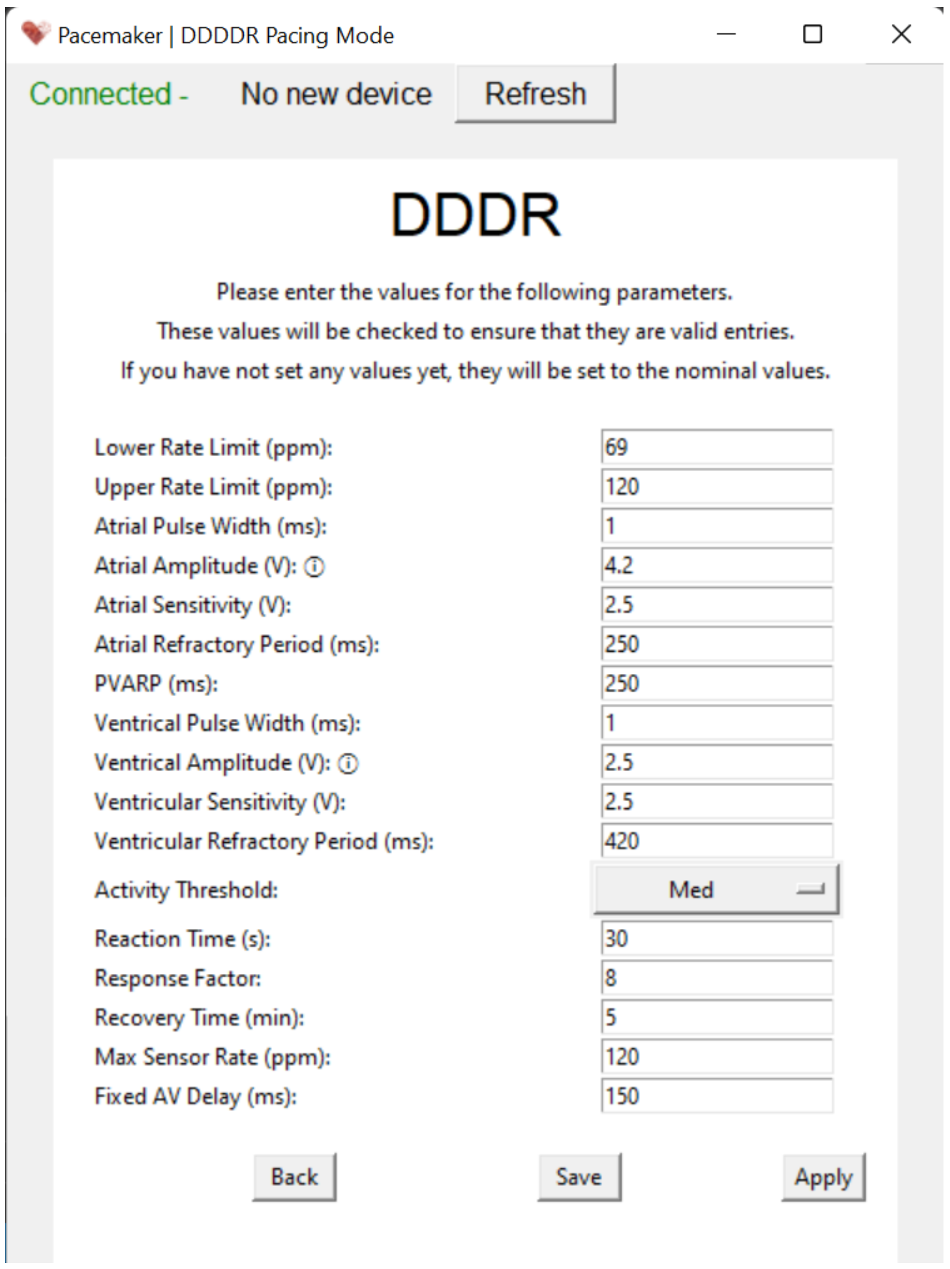
Demo

Some of our favourite pages!

Welcome Page	 Pacemaker Not connected - No new device Welcome
Login Screen	 Pacemaker Login Not connected - No new device Login

Registration Page	 Pacemaker Register Not connected - No new device Register Username Password Re-enter password Back Register

Pacing Mode Selection Page with the nominal value selected (DDD)	 Pacemaker | Mode Selection Connected - No new device Please choose a pacing mode AOO / AOOR VOO / VOOR AAI / AAIR VVI / VVIR [✓ DDD] / DDDR

DDDR Pacing Mode Page: initially with some values specified in "PACEMAKER" document	 This is just an example. All of the other pacing modes look similar, just with their respective parameters and nominal values initially inputted.

DDDR

Please enter the values for the following parameters.
These values will be checked to ensure that they are valid entries.
If you have not set any values yet, they will be set to the nominal values.

Lower Rate Limit (ppm):	69
Upper Rate Limit (ppm):	120
Atrial Pulse Width (ms):	1
Atrial Amplitude (V): ⓘ	4.2
Atrial Sensitivity (V):	2.5
Atrial Refractory Period (ms):	250
PVARP (ms):	250
Ventrical Pulse Width (ms):	1
Ventrical Amplitude (V): ⓘ	2.5
Ventricular Sensitivity (V):	2.5
Ventricular Refractory Period (ms):	420
Activity Threshold:	Med
Reaction Time (s):	30
Response Factor:	8
Recovery Time (min):	5
Max Sensor Rate (ppm):	120
Fixed AV Delay (ms):	150