Model-Based Conformance Testing for Implantable Pacemakers

George Major Chen, Biomedical Engineering, Johns Hopkins University, *SUNFEST Fellow*

Zhihao Jiang, Computer and Information Science, University of Pennsylvania, *Graduate Student*

Rahul Mangharam, Ph.D. Department of Electrical and Systems Engineering, University of Pennsylvania

***Abstract*— Between 1990 and 2000, over 600,000 implantable cardiac pacemakers and cardioverter defibrillators were recalled. 41% of these devices were recalled due to device software issues. Software-related recalls are increasing with the growing complexity of medical device software, which is responsible for the life-critical operation with the organ. Currently, there are no formal methods to test and verify the safety of implantable cardiac device software. To meet this need, a pacemaker-testing platform has been developed to automatically verify that the software in a pacemaker is functioning appropriately and determine if the pacemaker implementation conforms to the device software design specifications. A testing methodology was developed where tests were automatically generated from a model of the pacemaker that satisfied the specifications. These tests checked the software implemented in the physical pacemaker were in conformance with the design specifications and ensured safe operation. This paper outlines the steps used to create this testing platform, as well as the steps used to construct a pacemaker model for testing. By using this test framework as a standard for medical device testing, the US Food and Drug Administration (FDA) will potentially have a more streamlined method to certify the safety of medical device software.**

***Index Items*: Pacemakers, software validation, model-based development, safety analysis**

I. INTRODUCTION

Over the past few decades, implantable cardiac devices such as pacemakers have been widely used to treat arrhythmia, which are heart diseases featuring irregular heart rhythms. However, the number of life-threatening device malfunctions increases as the complexity of the device software increases. Between 1990 and 2000, over 600,000 implantable cardiac devices were recalled, and the percentage of medical device recalls due to *software-related issues* increased from 10% to 21% [1].

Currently there is no systematic way to evaluate the safety of pacemaker software. The Food and Drug Administration (FDA) certifies devices like pacemakers based on the extensive test reports submitted by device manufacturers [2]. The primary approach for system-level testing of pacemakers is unit testing, which requires playing a pre-recorded electrogram signal into the pacemaker and recording the output of the pacemaker. This helps to evaluate if the input signal triggered the appropriate response by the pacemaker, but has no means of evaluating if the response was appropriate for the patient condition [3]. Moreover, this *open-loop testing* method by the device manufacturers is not able to find potential safety violations that involve closed-loop interaction between the device and its environment (i.e. the heart or the patient).

The test cases used to evaluate pacemakers are also not systematically generated to guarantee coverage over all possible scenarios in the software specification. As a result, open-loop testing cannot guarantee the safety of the pacemaker software. Furthermore, with the patient in the loop, it is necessary to devise a new testing methodology, as all possible cases cannot be enumerated.

Previous efforts on the verification of time-critical and safety-critical embedded systems have been done [4]; however, these methods have only started to be implemented for medical device evaluation and verification. D. Arney, R. Jetley, P. Jones, I. Lee, and O. Sokolsky [5] have used Extended Finite State Machines for model checking of a resuscitation device. Additionally, formalized methods to improve medical device protocols [6] and safety [7] have been created. However, the authors either used a simplified patient model or did not use a patient model in their methods.

The focus of this paper is on the development of tools and methodologies to test and formally verify whether the software in pacemakers is safe within the closed-loop context of the patient. Section II provides a brief overview of heart electrophysiology, the operation of pacemakers, and timed automata modeling. Section III provides the Methodology for developing the tools. Section IV details UPPAAL timed automata modeling efforts. Section V describes how the UPPAAL Timed Automata Model was translated into a MATLAB Implementation. Section VI describes how the MATLAB Implementation was translated into a hardware implementation. Section VII concludes the work. Section VIII presents applications of the project. Section IX describes future work.

II. Background

A. Heart Anatomy and Electrophysiology

To maintain and regulate proper function, the heart generates electrical impulses which help to organize muscle contractions involved in pumping blood to the rest of the body. The heart consists of four chambers, the left and right atriums, which obtain blood from the body; and the left and right ventricles, which obtain blood from the left and right atriums respectively and pump blood to the body (Fig 1.). A tissue located on the right atrium, called the Sinoatrial (SA) node periodically self-depolarizes. This depolarization signal then travels to both atria, causing contractions which pushes blood into the ventricles. The signal is then delayed at the Atrioventricular (AV) node, which allows the ventricles to fill fully before being stimulated. The His-Purkinje system then spreads the signal to both ventricles, which causes contractions in the ventricles to push blood to the rest of the body. Any impairment or anomalies of this electrical system can cause heart arrhythmias, which affect the heart's ability to properly pump blood to the rest of the body [8].

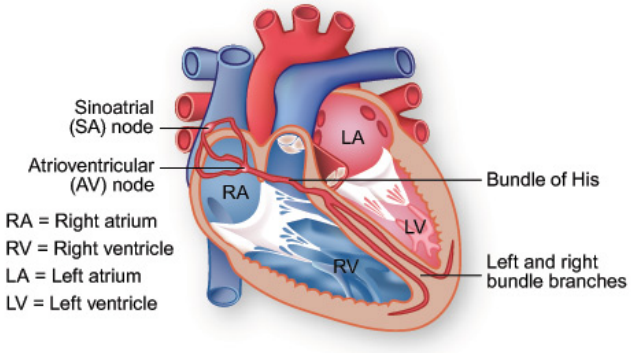


Fig 1. Heart Anatomy and Electrical Conduction System

B. Pacemaker Mechanics and Actuation

A pacemaker is an electronic device implanted into a patient to regulate his or her heart rhythm. It generally consists of a battery and electronic circuits sealed in a metal enclosure with leads. The number and the use of these leads are dependent on the pacemaker model. A DDD (*Dual* chamber sensing, *Dual* chamber pacing, and *Dual* mode of response) pacemaker has two leads attached to a patient's right atrium and right ventricle to sense the electrical activity of the heart and apply the appropriate stimulus to pace the heart if an arrhythmia is detected.

Common nomenclature for pacemaker activities include AS (atrial sense) if the pacemaker detected a signal from the atrium, VS (ventricular sense) if the pacemaker detected a signal from the ventricle, AR (atrial refractory) if the pacemaker detected a signal from the atrium during a refractory period, VR(ventricular refractory) if the pacemaker detected a signal from the ventricle during a refractory period, AP (atrial pace) if the pacemaker paced the atrium, and VP (ventricular pace) if the pacemaker paced the ventricle [9].

C. Electrogram

An intracardiac electrogram (EGM) is a recording of the potential differences between two electrodes on each lead of a pacemaker. In a DDD pacemaker, one lead senses the intra-atrial EGM; the other, the intra-ventricular EGM. The pacemaker uses these recordings to time appropriate events for pacing [9].

D. Pacemaker Timing Cycles

Fig. 2 presents an overview of the basic timing cycles of a DDD pacemaker. The five different timing cycles are denoted as LRI, URI, AVI, PVARP, and VRP.

The Lowest Rate Interval (LRI) timing cycle is initiated in response to ventricular events (VS, VP) and helps to prevent bradycardia, or a slower-than-normal heart rhythm. Depending on the algorithm, the pacemaker will deliver ventricular pacing if another ventricular event is not detected after the LRI, or will deliver atrial pacing if an atrial event (AS, AP) is not detected in the Atrial Escape Interval (AEI) which is initiated during a ventricular event and is equal to the length of the LRI minus the length of the AVI.

The Upper Rate Interval (URI) timing cycle is initiated in the same way as the LRI and helps to prevent tachycardia, or a higher-than-normal heart rhythm. If an atrial sense is detected early, the pacemaker will wait until the end of the URI period to deliver ventricular pacing.

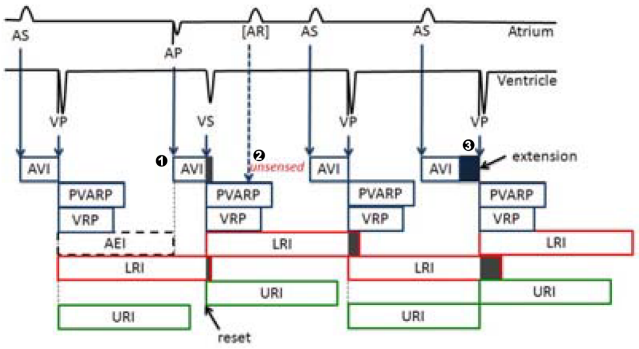


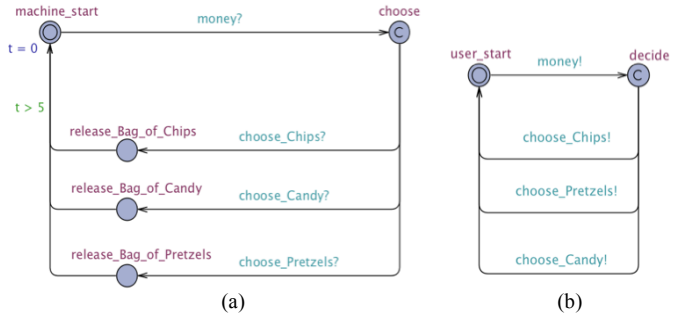
Fig. 2. Timing cycles of a DDD pacemaker [10]

E. Timed Automata Modeling

To best model the operations of a pacemaker, a timed automata is used. Timed automata are finite automata with a finite set of clocks [11]. Each state of a timed automata model is therefore not only event-based, but also time-based. This is commonly used for modeling systems that are triggered by time-based events. UPPAAL is a standard software tool used to help generate timed automata models, and to verify these models.

A common example of a timed automata model is a vending machine, shown in Fig. 3a. The vending machine stays in an idle state, defined by the node location "machine_start". If a user gives money, the vending machine changes state to the "choose" location. Depending on the user's choice, the vending machine can either release a bag of chips, bag of pretzels, or bag of candy. After roughly five seconds, the machine returns back to the idle state. The user can also be expressed as an automata model as well. Once the user gives money, the user changes state to the "decide" location. The user can then decide to choose chips, pretzels, or candy. Once

that decision is made, the user returns back to an idle state as well.



(a) (b)

Fig. 3. (a) A timed automata model of a typical vending machine operation. (b) A timed automata model of a typical vending machine user operation.

III. Methodology

A. Pacemaker Modeling

In [2], Pajic et.al proposed a *model-based design framework* for pacemaker software verification and testing. The pacemaker specification [3] provided by the device manufacturers was converted to a Timed Automata model representation. As the first step, the safety of the specification is evaluated by formally verifying the pacemaker model in closed-loop with a model of the heart.

In this project, we use *model-based conformance testing* to check whether a pacemaker software implementation has successfully implemented its specification. With proved safety of the specification and its rigorous implementation, the safety of the pacemaker software can be guaranteed.

Fig. 4 presents an overview of the steps used to produce a pacemaker model. In order to create a pacemaker model that can take in specifications from manufacturers, a UPPAAL Timed Automata model of the operations of a pacemaker is constructed with adjustable parameters. Once the Timed Automata model is verified, it is then converted into a MATLAB model for simulation. The MATLAB model is then tested to ensure that the pacemaker operations can be appropriately adjusted to meet device specification. Once complete, the MATLAB model is then implemented into a hardware platform, which simulates in real-time the operations of a pacemaker according to specification.

B. Testing Framework

The testing framework (Fig. 5) consists of a *Test Generator* and a *Test Platform*. Given a pacemaker specification in timed automata representation, the Test Generator generates a series of test cases so that the executions of the test cases satisfy certain coverage criteria. Each test case consists of a sequence of inputs and expected outputs. The test platform executes the test case by sending the inputs to the pacemaker implementation and compares its output with the expected output specified in the test case. If the pacemaker implementation passes all the test cases, we conclude that the pacemaker conforms to its specification.

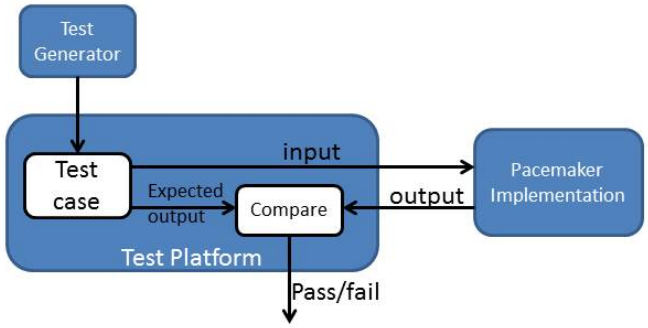


Fig. 5 Testing framework

IV. UPPAAL Model Specifications

A. Previous Work

In [12], Jiang et.al have developed timed automata models of pacemakers in UPPAAL, which have closely followed the timing cycles specified in Section II. This however, presents problems with scaling the model to all pacemaker specifications. Due to the proprietary nature of pacemaker software, it is difficult to determine how the specific timing operations of any pacemaker work. Pacemakers may use more timers specified in Section II and perform more complex operations. Since these factors are unknown, testing these pacemakers is therefore black box; there is no means of determining internally in software how the pacemaker operates. The model that serves as a ground truth for these tests must then take these considerations into account. In order to create a model that can incorporate all pacemaker operations the authors propose a two-clock timed automata model.

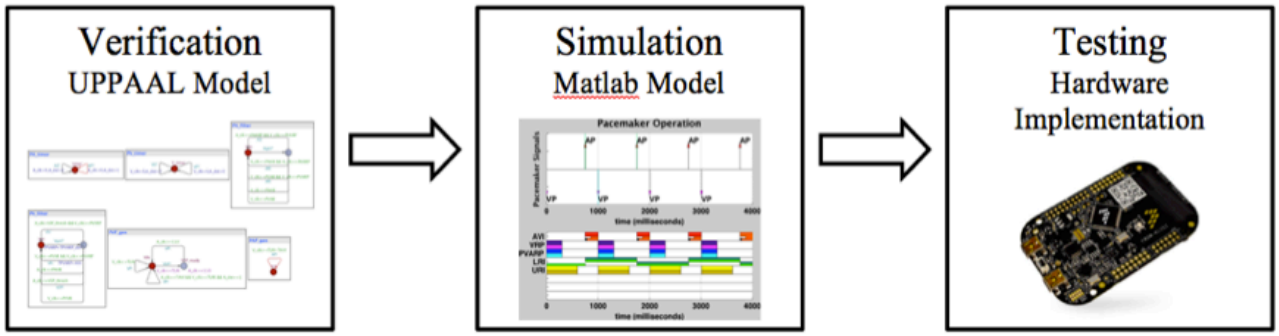


Fig. 4. Pacemaker Modeling Framework

B. *Two Clock Model*

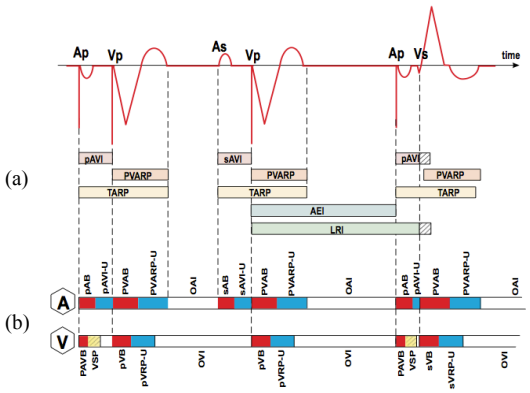


Fig. 6 (a) A multiple clock pacemaker operation model. (b) A two clock pacemaker operation model. [9]

A two-clock timed-automata model of a pacemaker (Fig. 6(b)) uses a clock that times ventricular events (VS, VR, VP) and another that times atrial events (AS, AR, AP). By removing multiple timers and basing operating decisions on only two timers, the model increases its scalability of incorporating different pacemaker models. Furthermore, the basic timing cycles of a pacemaker are also retained. Instead of resetting different timers intermittently and turning them on and off, the model represents the basic timing cycles by using the current clock time and comparing it to an expected timing cycle.

Table I lists the notations used in the model and Figs. 7,8,9 present the clock operations, the chamber event detection logic, and the pacing decisions respectively. In all instances, when an atrial clock begins to time, the beginning of the timer is the PAAB, PAARP and TAVI period, then the PVAB and the PVARP period. Similarly, when the ventricular clock begins to time, the beginning of the timer is the PVVB and PVVRP period, then the PAVB, PAVRP, and VSP period. In both cases, the TURI and TLRI are checked to ensure that the heart is not beating too slowly or too quickly.

Notation	Definition	Notation	Definition
A_clk	The current time of the atrial clock	PAAB	Post-Atrial Atrial Blocking period, a period in the atrial clock right after an atrial event that blocks any atrial stimulus.
V_clk	The current time of the ventricular clock	PAVB	Post-Atrial Ventricular Blocking period, a period in the atrial clock right after an atrial event that blocks any ventricular stimulus.
A_det	"If atrial event is detected". A Boolean to determine if an atrial event occurred before a ventricular event.	PVAB	Post-Ventricular Atrial Blocking period, a period in the ventricular clock right after a ventricular event that blocks any atrial stimulus.
V_det	"If ventricular event detected". A counter that counts the amount of times a ventricular event occurs before an atrial event.	PVVB	Post-Ventricular Ventricular Blocking period, a period in the ventricular clock right after a ventricular event that blocks any ventricular stimulus.
VSP	Ventricular Safety Pacing. A Boolean that determines if ventricular safety pacing should be applied.	PAARP	Post-Atrial Atrial Refractory Period, a period in the atrial clock right after an atrial event that determines if an atrial event is defined as AR.
Aget	If atrial event was detected. A Boolean that determines if an atrial event was detected	PAVRP	Post-Atrial Ventricular Refractory Period, a period in the atrial clock right after an atrial event that determines if a ventricular event is defined as VR.
Vget	If ventricular event was detected. A Boolean that determines if ventricular event was detected	PVARP	Post-Ventricular Atrial Refractory Period, a period in the ventricular clock right after a ventricular event that determines if an atrial event is defined as AR.
AP	Atrial Pacing event	PVARP_def	The defined time for a Post-Ventricular Atrial Refractory Period
AS	Atrial Sensing event	VSP_thresh	Ventricular safety pacing threshold period, a period in the atrial clock right after an atrial event that if enabled, will cause the pacemaker to send a VP after this period.
AR	Atrial Refractory event	TLRI	Total Lower Rate Interval Time, the longest time any atrial-to-atrial or ventricular-to-ventricular event can be.
VP	Ventricular Pacing event	TURI	Total Upper Rate Interval Time, the shortest any atrial-to-atrial or ventricular-to-ventricular event can be.
VS	Ventricular Sensing event	TAVI	Total Atrial Ventricular Interval Time, the period between any atrial-to-ventricular event.
VR	Ventricular Refractory event	TLRI-TAVI	A derived clock time using TLRI and TAVI. Represents the Atrial Escape Interval, or the period between any ventricular-to-atrial event.

Table I. List of notations used for this model.

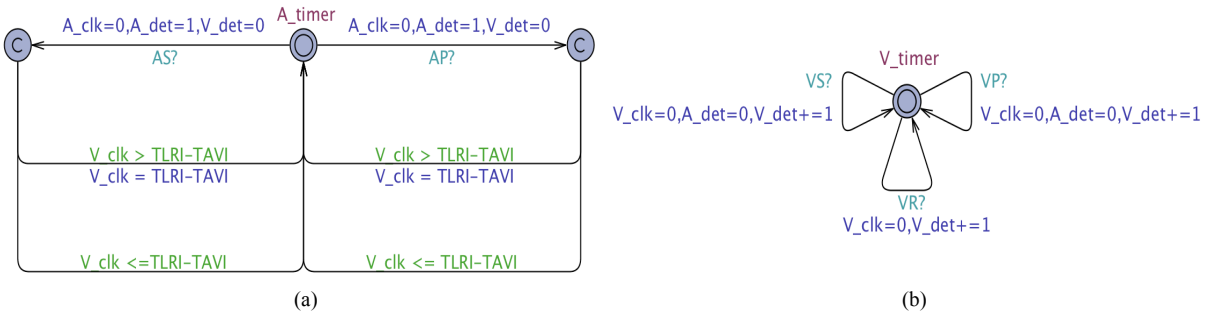


Fig. 7 (a) Atrial timer model. (b) Ventricular timer model.

The atrial clock (Fig. 7(a)) is reset when an AS or AP is detected. Additionally, A_det is turned true, and V_det is reset. To ensure AV synchrony, if the ventricular clock is greater than the Atrial Escape Interval, the ventricular clock is reset back the total Atrial Escape Interval Time.

The ventricular clock (Fig. 7(b)) is reset when a VS, VP, or VR is detected. Additionally, A_det is turned false, and V_det increments 1 to count the ventricular event.

For atrial event detection (Fig. 8(a)), if Aget is true the model enters a state that decides how the event will be recognized as. The pacemaker detects the atrial event as AS if the atrial clock time is past PAARP and if the ventricular clock time is past PVARP. The pacemaker detects the atrial event as AR if the atrial clock time is past PAAB and is in PAARP, or if the ventricular clock time is past PVAB and is in PVARP. If the atrial clock is in PAAB, or if the ventricular clock in PVAB, the atrial event is ignored.

For ventricular event detection (Fig. 8(b)), if Vget is true the model enters a state that decides how the event will be recognized as. Additionally, PVARP is reset to PVARP_def, if in case the value of PVARP is changed. If the atrial clock time is in the VSP_thresh and is past PAVB, the pacemaker ignores the event, and enables VSP. If the atrial clock is past VSP_thresh and if the ventricular clock is past PVVRP, the pacemaker detects the ventricular event as VS. Before returning back to the initial idle state, the model evaluates a few more cases. If V_det equals 0, the model returns back to the idle state. Otherwise, if V_det is greater than or equal to 1, the model checks if PVARP_def is greater than or equal to 400. If it is, the model returns back to the initial state. Otherwise, it assigns PVARP to 400. If the ventricular clock is past PVVB and is in PVVRP, the pacemaker recognizes the event as VR. If PVARP_def is greater than or equal to 400, the model will return back to the idle state; otherwise, it will assign PVARP to 400. If the atrial clock is past PAVB, or if the ventricular clock is in PVVB, the event is ignored.

For atrial pacing (Fig. 9(a)), the model simply provides pacing if the atrial clock is greater than or equal to TLRI.

For ventricular pacing (Fig. 9(b)), the model provides pacing if the ventricular clock is greater than or equal to TLRI, or if all of the conditions are met: 1) the atrial clock is greater than or equal to TAVI, 2) the ventricular clock is greater than or equal to TURI, 3) A_det is true, and 4) V_det equals zero. If VSP is enabled, the ventricular pacing model goes to the VSP_ready state. Once the atrial clock time is greater than VSP_thresh, the model provides pacing.

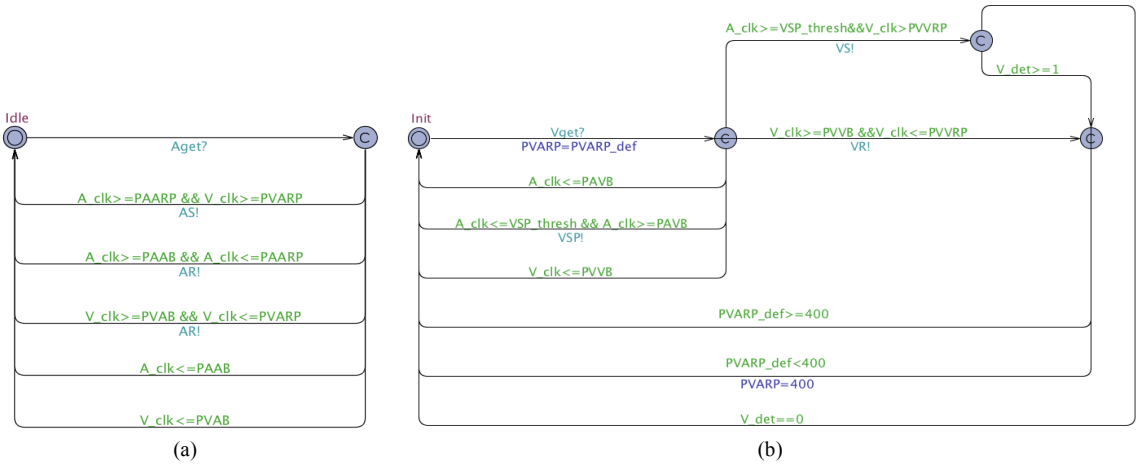


Fig. 8 (a) Atrial event detection model. (b) Ventricular event detection model

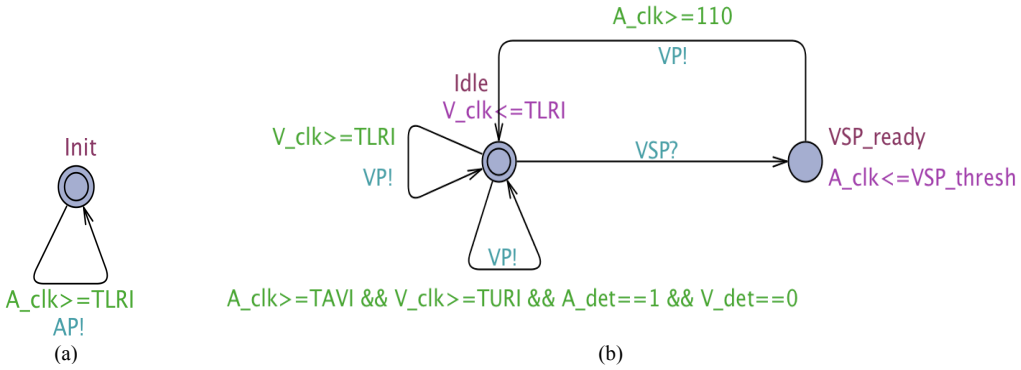


Fig. 9 (a) Atrial Pacing Model. (b) Ventricular Pacing Model.

C. *Results and Discussion*

This UPPAAL model now serves as the starting point for creating real-time system model. Due to its decreased reliance on multiple timers, this pacemaker model is more capable of implementing more specifications. Additionally, the major timing cycles can be adjusted and changed in this model to meet general pacemaker specifications. Furthermore, UPPAAL is also capable of outputting the operation of the model in a symbolic trace format, which can later be used to help generate test cases.

V. MATLAB IMPLEMENTATION

A. *Motivation*

Though UPPAAL is a versatile tool for constructing and verifying timed automata models, there are some limitations. The timed-automata model simulator in UPPAAL expresses the changes in the model in a state-to-state format. This makes it difficult to visualize the changes in the model in real time, and to provide error correction if the model changed to a specific state at an inappropriate time.

Furthermore, for this project, the model must be able to act as a ground-truth to test if a pacemaker is functioning correctly. Therefore, the model must be in a format that is easy to compare against any pacemaker. Since the exact pacemaker software is not known, it is difficult to express a specific pacemaker in a UPPAAL format to make it comparable to the UPPAAL model. UPPAAL is also not able to check and compare two timed automata models and provide quantifiable data on the differences.

The authors of this paper implemented the UPPAAL model into MATLAB. Because of UPPAAL's code structure, the model is very easily translatable, and MATLAB provides many libraries to help simulate and visualize the model in a time-step format. Additionally, a virtual tester can be implemented on MATLAB to test the model against test cases.

B. *Tester*

i. Test Case

Fig. 10 presents the file format for test cases. The left column indicates time in milliseconds of an expected event, and the right column indicates the type of event. The numbers in the second column represent the following:

- 1: Atrial Input given to the pacemaker
- 2: Ventricular Input given to the pacemaker
- 3: Expected atrial pacing from the pacemaker
- 4: Expected ventricular pacing form the pacemaker

These test files are fed into the tester to help inform when stimuli should be applied to the pacemaker and when pacing events should occur.

	1	2
1	20	2
2	30	1
3	40	2
4	60	1
5	130	2
6	200	2
7	1000	3
8	1020	1
9	1030	2
10	1070	1
11	1110	4
12	1450	1
13	1700	4

Fig. 10. Test Case File.

ii. Algorithm

Since testing the pacemaker is black box and the current state of the pacemaker is unknown, prior to testing, the pacemaker needs to be set into a known state. This helps to ensure that the test was performed when the pacemaker was operating appropriately.

To set the pacemaker in a state that is known for testing, an initializer file (Fig. 11) specific to the specification is entered into the tester. The initializer file is in the same format as a test file, but consists only of atrial inputs and ventricular inputs to simulate a natural heart rhythm.

Once the pacemaker is initialized, the tester reads in the test-case file line by line. Additional parameters, such as the acceptable range of error for the proper time for pacing can be implemented into the tester algorithm. Fig. 12 presents some sample output of the tester algorithm.

Once the current test case is complete, the tester reads the next test case, and evaluates. If the next test case is a sensing event, the tester will wait until that time to give the appropriate sensing signal.

	1	2
1	200	1
2	450	2
3	1200	1
4	1450	2
5	2200	1
6	2450	2

Fig. 11. A sample initializer file.

If an atrial pace or ventricular pace is detected at a time not specified by the current test case, the tester states the error and ends the test. Errors that can be reported are early, late, or unexpected pacing. The tester algorithm evaluates the reason for the error by first checking if the pacing was expected in the current test case, but not within the expected time.

```
initializing...

starting test
Pacemaker paced ventricle on time at t=0. (Expected at t=0. Misalignment: 0)
sent ventrical signal at t=50.
sent atrial signal at t=200.
Pacemaker paced atrium on time at t=750. (Expected at t=750. Misalignment: 0)
Pacemaker paced ventricle on time at t=1001. (Expected at t=1000. Misalignment: 1)
```

```
initializing...

starting test
Pacemaker paced ventricle on time at t=0. (Expected at t=0. Misalignment: 0)
sent ventrical signal at t=300.
ERROR: Pacemaker incorrectly paced atrium at t=800.
```

Fig. 12. Sample output results of tester.

If the pacing was not expected with the current test case (i.e. the current test case expected an atrial pace, but a ventricular pace was detected), the tester reads the test cases immediately before and after the current test case, and evaluates if pacing was specified in said test cases. If those test cases did not specify a pacing event, the error is defined as 'unexpected pacing'. Likewise, if the next test case specified the pacing event, the error is defined as 'early pacing'. If the previous test case specified the pacing event, the error is defined as 'late pacing'.

C. *MATLAB Model*

With the tester, the MATLAB pacemaker model was evaluated using some initial Medtronic tests for a DDD pacemaker [13]. Shown are 2 of the 75 tests that were done (Fig. 13, Fig. 14). A green highlight indicates that the pacemaker properly paced the atrium at the right time; a blue highlight indicates that the pacemaker properly paced the ventricle at the right time; and a red highlight indicates that the pacemaker paced at the incorrect time. Of the 75 tests, the pacemaker model passed 72, thereby passing 96% of the tests.

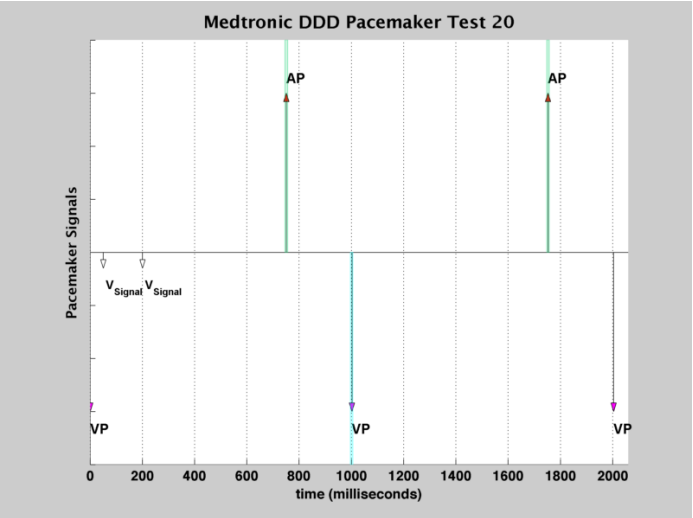


Fig. 13. A Medtronic test that the pacemaker passed.

D. *Results and Discussion*

The current pacemaker model has been tested against some Medtronic tests. The initial results show that the model can represent most tests. Some additional adjustments of the model can be made in UPPAAL or MATLAB and evaluated again.

Additionally, testing the pacemaker also served to evaluate if the tester algorithm is capable of determining errors, and if it is capable of determining if the pacemaker model passed the tests. Since the algorithm has been proven to work, this algorithm can be implemented in hardware. The next section discusses the efforts to transfer the current tester algorithm and pacemaker model into hardware.

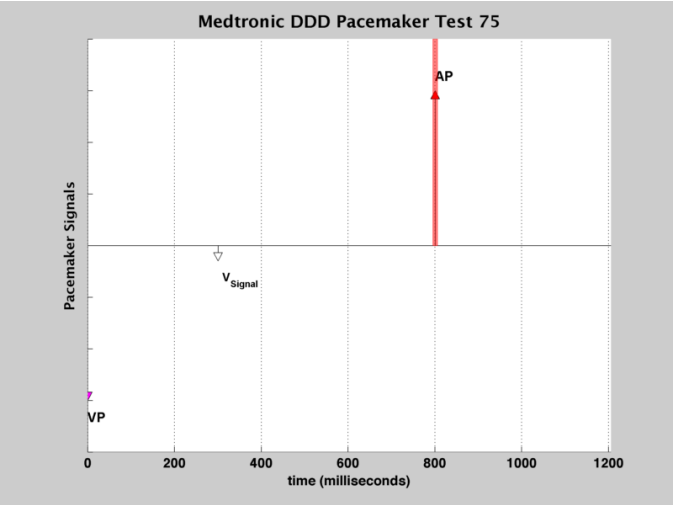


Fig. 14. A Medtronic test that the pacemaker failed

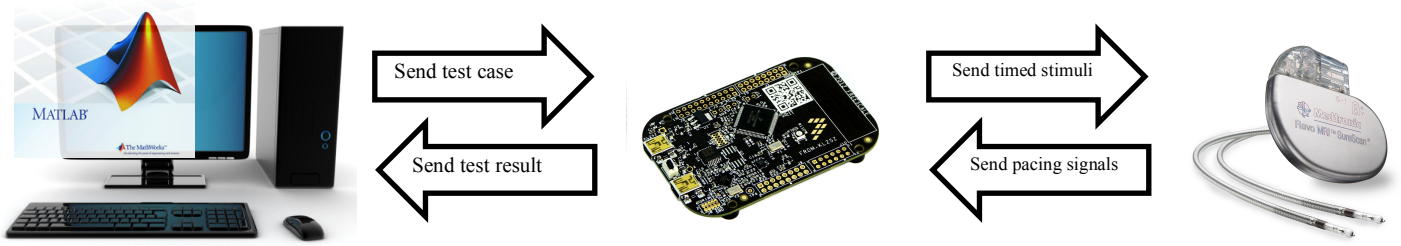


Fig. 15. Pacemaker Tester Setup.

VI. Harware Implementation

A. Architecture

A hardware implementation that can read in pacemaker signals and send out stimuli is needed in order to interface with a pacemaker to conduct tests. To increase user-friendliness a graphic user interface should make it simple for users to generate a series of test cases that meet specific criteria, send the cases to the tester, and then analyze the test results.

To meet these design goals, we propose the following test setup (Fig. 15):

- 1) A MATLAB graphic user interface that allows for users to set up and perform different tests to the pacemaker.
- 2) A microcontroller that can connect to the computer via USB and be capable of receiving information from MATLAB about the test, and perform it on a pacemaker.

B. Development

Previous work by Jiang et.al [8] has used FPGAs to implement system models of medical devices into a real-time setting. Though there are many benefits of using FPGAs for real-time systems, such as faster processing speed, FPGAs can get costly, are difficult to program, and may not be cost-effective relative to the task needed to be completed.

Since the tester is not very computational intensive, we decided to use the FRDM-KL25Z board as the hardware platform for the pacemaker tester. The FRDM KL25Z (Fig. 16) is a 48 MHZ, 32-bit ARM Cortex microcontroller with a serial USB interface. The tester detects pacemaker signals using digital interrupts, and produces simulated atrium and ventricle signals using GPIO pins.

Since the system is real-time, some protocols for the tester algorithm were changed to make the system operate more efficiently. The changes are as follows:

- To reduce the amount of real-time computations needed for the tester hardware to operate, reporting of information is done at the end of the test instead of real time. This helps to reduce the amount of time needed to send data back to the computer.
- Instead of sending the tester hardware the test case files line-by-line, whole, single files are sent to the hardware. This helps to further reduce the time needed to send data between the computer and tester board.
- Error reporting is done at the end of the test, and through MATLAB. The tester board sends to the computer a timed recording of the pacemaker operations. A cross comparison between the expected operation and the actual operation is then done in MATLAB to evaluate if the pacemaker passed the test or not.

Fig. 16. FRDM-KL25Z board

C. Pacemaker Implementation

To help in the process of testing and evaluating the tester hardware, the MATLAB pacemaker model was implemented into a FRDM-KL25Z board. Fig. 17 shows typical operation of the pacemaker hardware model when not given stimulus. The hardware implementation of the pacemaker uses two GPIO pins to simulate atrium and ventricle pacing, and two interrupts to simulate the detection of atrium and ventricle stimulus respectively. The device provides pacing at 60 beats per minute when no atrial or ventricular events are detected.

Fig. 17. Pacemaker hardware operation given no stimulus. (a) Ventricle pacing provided by the pacemaker. (b) Atrium pacing provided by the pacemaker. (c) Overall operation. Ventricle-to-ventricle pacing time and atrium-to-atrium pacing time were 1000 ms (60 beats/min).

D. Matlab Graphic User Interface

To increase usability of the system, a graphic user interface was created (Fig. 18). The interface allows for the user to import test case files and select which tests of those tests to perform. Different test parameters can be adjusted, and the user can also select whether to run the test against a simulation, or with the hardware tester. The interface also allows the user to change the timing operations of the pacemaker implementation if the pacemaker implementation is used. After each test is performed, the interface displays the visual results, as well as the results in text.

E. Testing

Some initial testing was performed to evaluate the efficiency of the hardware tester software. Serial communication protocols between the tester hardware and the MATLAB interface were evaluated to determine the speed of transmitting data. The full platform (Fig. 19) has both the tester and pacemaker implementation communicating with each other, and results of the tests are transmitted back to the computer interface for analysis.

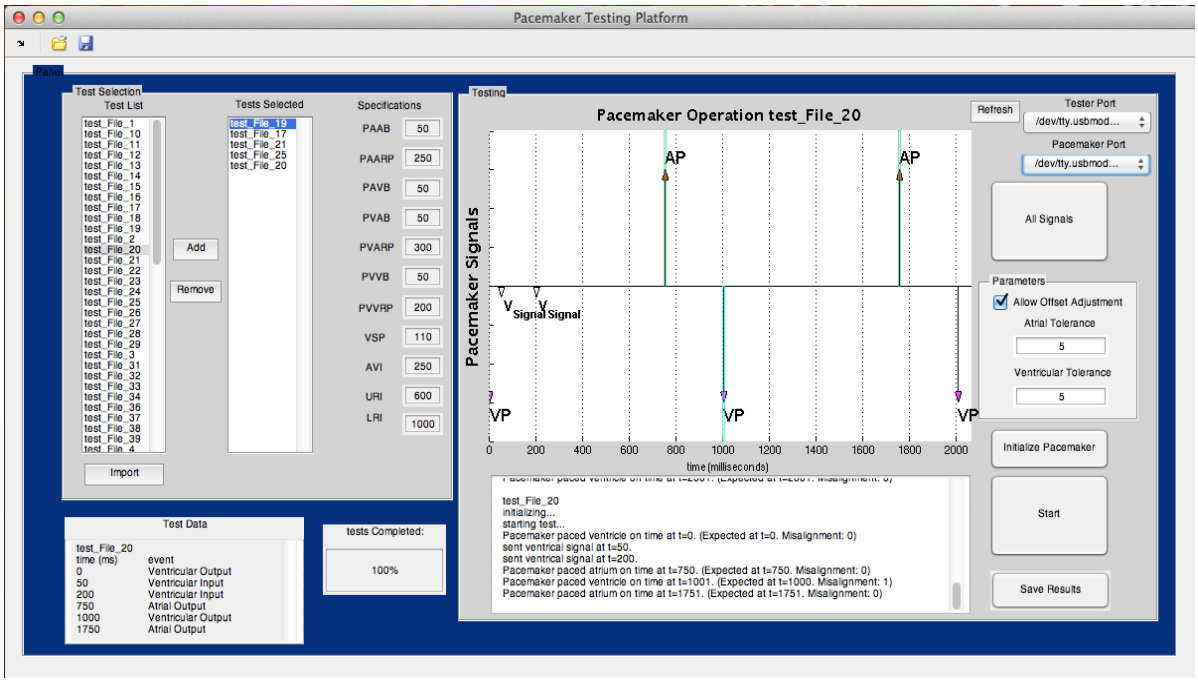


Fig. 18. Matlab graphic user interface for the pacemaker testing platform

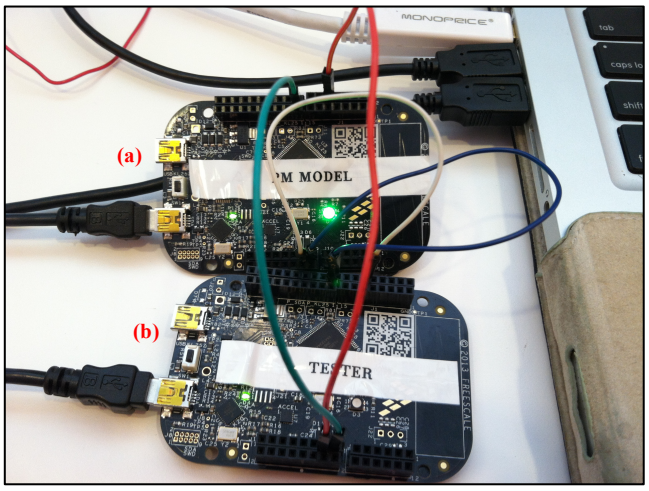


Fig. 19. The full tester platform. (a) The pacemaker model implemented onto a board. (b) The tester board that interfaces with MATLAB.

VII. Conclusion

Using a pacemaker timed automata model in UPPAAL, a hardware implementation was constructed in addition to a tester to evaluate the operation of a pacemaker. The current version of the pacemaker-testing platform reads in a test-case file and evaluates the pacemaker depending on the expected events given by the test case file. So far, 75 tests from Medtronic have been used to test the pacemaker model and have been executed using the testing platform. To increase the number of tests that can be performed, later iterations of this testing platform will be able to use the pacemaker model to generate multiple different tests.

The pacemaker model that was constructed can take in different types of timing parameters. Therefore, the model can be used to predict the expected operations of different pacemakers. Since the current pacemaker model is based on a typical DDD pacemaker operation, later iterations of this model will be able to also adjust the algorithms used to regulate the heart.

A testing platform using FRDM-KL25Z microcontrollers has been also created which will serve as an interface to test different pacemakers. Evaluation and analysis of the pacemaker operation can also be recorded and done through MATLAB.

In conclusion, the pacemaker model and testing platform presented in this paper provide a framework to create a robust testing platform for pacemakers. A complete testing platform will provide medical device companies a tool to properly evaluate pacemaker software. It will also provide the FDA with a streamlined method to validate and certify pacemakers before allowing them to go on the market.

VIII. Future Work

A. Pacemaker Modeling

i. More pacemaker mode algorithms

Pacemakers can function in different ways depending on the mode. Generally speaking, pacemakers can sense, pace, and respond to exclusively the atrium, ventricle, or both respectively. The current pacemaker model implemented is a DDD pacemaker, which allows for the sensing, pacing, and responding of both chambers. To increase the robustness of the pacemaker model, these modes should be implemented, which will then further help to incorporate more pacemakers into the testing platform.

ii. Increase coverage

Additional algorithms used in pacemakers to treat specific heart arrhythmias, such as endless loop tachycardia, are different for each pacemaker. However, the overall expected outcome is the same. These additional algorithms should also be incorporated into the pacemaker model.

B. Testing Algorithms

i. Quantification of test coverage

Though an exhaustive evaluation of a pacemaker can be done with a series of different test cases, the process is time-consuming, and even still, may leave out certain cases. Quantification of overall coverage of series of tests can help to determine if additional tests need to be performed to determine if a pacemaker's software is working within safe conditions, or if a selective series of tests instead of all can be performed for maximum coverage. The algorithms needed to quantify the coverage of test cases should be implemented in later iterations of the pacemaker-testing platform.

ii. Symbolic to Concrete Trace

UPPAAL is capable of simulating the changes of a timed-automata model in an event-based format. This symbolic representation of the changes in the model can be converted into a concrete format, where the exact times of the changes can be recorded. By implementing this technique with the UPPAAL pacemaker model, test cases can be quickly and easily generated. Since the simulation can last as long as needed, tests can also be varied in overall time, which can further help to do more extensive testing for longer periods of time.

iii. Initialization sequence

In order to set a pacemaker into a known state for testing, an initialization sequence is run which simulates a regular heart rhythm. Increase the robustness of the initialization sequence, algorithms can be constructed to test if the pacemaker being tested is in a proper state after given an initialization sequence. The initialization sequence can then be adjusted to different pacemakers if needed.

IX. Applications

A. Pacemaker Development and Testing

The pacemaker-testing platform can be used by medical device companies to help develop and test new pacemaker software. Since the platform is standardized, companies can perform the same tests to determine if the software they have developed is operating within safe conditions. Since the testing platform automates the testing process, using the platform can also aid in rapid development, and faster feedback of results.

B. FDA Test Approval

The pacemaker-testing platform can be used by the FDA as a standard assessment to test, validate, and verify pacemakers before they get released on the market. Currently, the FDA certifies implantable cardiac devices based on extensive test reports provided by the manufacturers. By using the pacemaker-testing platform, the FDA can have a formal method to approve pacemakers, which can also help in providing companies faster approval or disapproval.

Acknowledgment

The authors would like to thank all members of mLab for their support and collaboration on the project. The authors would also like to thank Abhijeet Mulay for his work on the hardware aspects of the project.

The authors would also like to acknowledge the support of the National Science Foundation, through NSF REU grant no. 1062672.

References

[1] "List of Device Recalls, U.S. Food and Drug Admin. (last visited Jul. 19, 2012)."

[2] K. Sandler, L. Ohrstrom, L. Moy, and R. McVay, "Killed by Code: Software Transparency in Implantable Medical Devices," *Software Freedom Law Center*, 2010.

[3] J. M. Cortner, "Testing Implantable Medical Devices," *Global Healthcare Medical Device Manufacturing Technology*, pp. 2–4, 2003.

[4] E. M. Clarke and J. M. Wing, "Formal Methods: State of the Art and Future Directions," *ACM Computing Surveys*, vol. 28, pp. 626–643, 1996.

[5] D. Arney, R. Jetley, P. Jones, I. Lee, and O. Sokolsky, "Formal Methods Based Development of a PCA Infusion Pump Reference Model: Generic Infusion Pump (GIP) Project," in *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSSMDPnP. Joint Workshop on*, 2007, pp. 23 –33.

[6] R. Alur, D. Arney, E. L. Gunter, I. Lee, J. Lee, W. Nam, F. Pearce, S. Van Albert, and J. Zhou, "Formal Specifications and Analysis of the Computer-Assisted Resuscitation Algorithm (CARA) Infusion Pump Control System," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 5, pp. 308–319, 2004.

[7] A. ten Teije, M. Marcos, M. Balser, J. van Croonenborg, C. Duelli, F. van Harmelen, P. Lucas, S. Miksch, W. Reif, K. Rosenbrand, and A. Seyfang, "Improving medical protocols by formal methods," *Artificial Intelligence in Medicine*, vol. 36, no. 3, pp. 193 – 209, 2006.

[8] Z. Jiang, M. Pajic, and R. Mangharam. "Cyber-Physical Modeling of Implantable Cardiac Medical Devices," *Proceedings of the IEEE,* pp.122-137, 2012.

[9] S. Barold, R. Stroobandt, and A. Sinnaeve. Cardiac Pacemakers Step by Step. Blackwell Future, 2004.

[10] Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam,"Real-Time Heart Model for Implantable Cardiac Device Validation and Verification," in 22nd Euromicro Conference on Real-Time Systems (ECRTS), July 2010, pp. 239 –248.

[11] R. Alur and D. L. Dill. A Theory of Timed Automata. Theoretical Computer Science, 126:183–235, 1994.

[12] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R.Mangharam, "Closed-loop Verification of Medical Devices with Model Abstraction and Refinement (submitted)," International Journal on Software Tools for Technology Transfer, 2013.

[13] DDD Pacemaker tests from Medtronic.