EPiC Series in Computing

Volume 90, 2022, Pages 18–25

Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)

Two Simulink Models with Requirements for a Simple Controller of a Pacemaker Device

Mostafa Ayesh, Namya Mehan, Ethan Dhanraj, Abdul El-Rahwan, Simon Emil Opalka, Tony Fan, Akil Hamilton, Akshay Mathews Jacob, Rahul Anthony Sundarrajan, Bryan Widjaja, and Claudio Menghi

McMaster University, Hamilton, Ontario, Canada
{ayeshm, mehann1, dhanraje, elrahwaa, opalkas, fant6, hamila10, jacoba11, sundarrr, widjajar, menghic}@mcmaster.ca

Abstract

This paper proposes a benchmark for a controller of a pacemaker device developed as part of the course "SFWRENG 3MD3 - Safe Software-Intensive Medical Devices" provided at McMaster University. The benchmark includes two alternative Simulink® models, developed by two different groups of students. Each model comes with a requirement formalized in Signal Temporal Logic (STL). We also present the testing results obtained using S-TaLiRo, a well-known testing framework for Simulink® models.

1 Introduction

This paper proposes a benchmark containing models with requirements for a simple controller of a pacemaker device. The models and requirements were developed as part of the course "SFWRENG 3MD3 - Safe Software-Intensive Medical Devices", provided at McMaster University (Canada). The course 3MD3 teaches students how to design, implement, verify and validate safe software-intensive devices and specifically medical devices. The elective course, offered in the 2021-2022 academic year, consisted of 24 students organized into groups. Each group had to design, implement, test, and deploy a simple controller for a pacemaker device (see Section 2). The students were supervised by a teaching assistant and a professor.[1] Two groups decided to submit their models and requirements as a part of this benchmark.

The benchmark contains the models, requirements, and testing results provided by the students. Specifically, the benchmark contains two models for the pacemaker controller defined using Simulink® [2], a widely used modeling language in the industrial domain. Each model comes with a requirement formalized in Signal Temporal Logic (STL) [9], a logic-based specification language for system requirements. The testing results are obtained by using S-TaLiRo [3], a well-known tool for falsification-based testing of Simulink® models.

[1] Respectively the first and the last author of the paper.

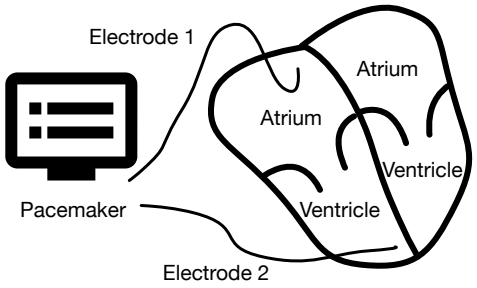


Figure 1: High-level schema of the pacemaker connections.

This paper (a) provides a high-level description of the behavior of the pacemaker controller, (b) presents the benchmark models and requirements, (c) summarizes the testing results, and (d) thoroughly discusses the results and outlines possible use cases for the benchmark. The benchmark and results are made publicly available [8].

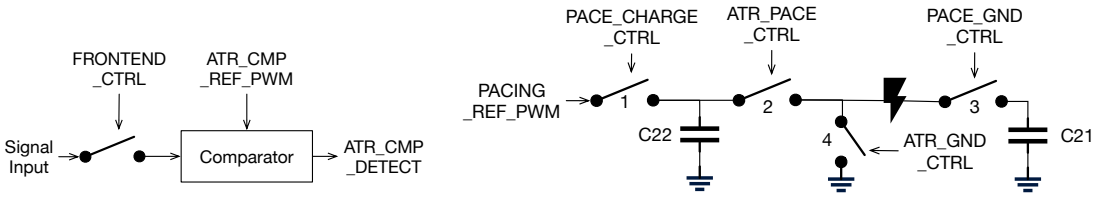
The paper is organized as follows. Section 2 presents a high-level description of functionality the pacemaker controller has to provide. Section 3 presents the Simulink® models of the controller proposed by the two groups of students. Section 4 describes the requirement proposed for each model and its encoding in STL. Section 5 describes the results of the testing activities. Section 6 discusses the lessons learned and outlines the possible use cases of the benchmark. Section 7 presents our conclusions and future work.

2 An Overview of the Pacemaker Controller

A pacemaker is a device that regulates the heart rate. To regulate the heart rate, the pacemaker is physically connected to the heart using one or more leads that are inserted into either the atrium, the ventricle or both. Figure 1 provides a high-level schema describing how the pacemaker is connected to the heart. The leads are used by the pacemaker to deliver electrical signals to the heart as well as monitoring the internal electrical activity of the heart. The pacemaker artificially stimulates the heart muscle to contract when no natural activity is present for a given time. Specifically, the pacemaker senses the heart to detect whether there is a natural activity in the heart. If no natural activity is detected for a given time, the pacemaker has to send electrical signals into the leads to force the heart to contract.

The controller senses and acts on the heart by controlling the sensing and the pacing circuits. In the following we present the sensing and pacing circuits for the atrium. The circuits associated with the ventricle are identical. The Simulink® models describing these circuits of the controller of the pace-maker were deployed and tested on a FRDM-K64F microcontroller.

The Sensing Circuit. The controller should use the sensing circuit to monitor the natural activity of the heart. Figure 2a presents a simplified schema for the sensing circuit. The inputs of the circuit are the signals `FRONTEND_CTRL` and `ATR_CMP_REF_PWM`. The signal `FRONTEND_CTRL` is used to control the switch that enables the sensing activity. The signal `ATR_CMP_REF_PWM` is the input provided to a comparator. The value of this signal is used as a threshold to detect whether there is a natural activity in the heart. The output of the circuit is the signal `ATR_CMP_DETECT`. Its value is high when the amplitude of the signal from the heart is above the threshold specified by the signal `ATR_CMP_REF_PWM`, the value is low otherwise.



(a) Sensing circuit.

(b) Pacing circuit.

Figure 2: High-level schema of the sensing and pacing circuits.

The Pacing Circuit. Figure 2b presents a high-level representation of the pacing circuit that is responsible for sending electrical signals to the heart. The circuit contains two capacitors (`C21` and `C22`) and four switches. The inputs of the circuit are the `PACING_REF_PWM` (determines the output signal's voltage 0-5V) and the values of the signals `PACE_CHARGE_CTRL`, `ATR_PACE_CTRL`, `ATR_GND_CTRL`, `PACE_GND_CTRL` controlling the status of the four switches. The output of the circuit is the signal `ATR_RING_OUT` which is sent to the lead inserted in the atrium.

The pacing circuit enables to pace the atrium by following two steps: the *preparation* and the *pacing*.

The *preparation step* prepares the circuit for the pacing activity. It requires to charge the capacitor `C22` and to discharge the capacitor `C21`. For this reason, switches 1, 3, and 4 must be closed, and switch 2 must be open: if switch 1 is closed and switch 2 is open, capacitor `C22` charges, if switches 4 and 3 are closed, capacitor `C21` discharges. Discharging the capacitor `C21` also enables removing residual charges present in the tissues of the heart.

The *pacing step* paces the heart. The pacing step requires to discharge the capacitor `C22` and charge the capacitor `C21`. For this reason, switches 2 and 3 must be closed and switches 1 and 4 must be open. Since switches 2 and 3 are closed and switches 1 and 4 must be open the current flows from capacitor `C22` to capacitor `C21` through the heart, that is the `ATR_RING_OUT` signal stimulates the desired chamber to contract .

The Pacemaker Controller. The pacemaker controller should work in different modes. In this paper, we describe the expected behavior of the controller for the AAI mode.

In the AAI mode, the controller should pace the atrium (A), sense the atrium (A), and remain inhibited (I) if the pacemaker senses natural activity. Figure 3 exemplifies the expected behavior of the controller in the AAI mode. The controller should monitor natural activity. If no natural activity is detected for a period of time (`Period`), the controller should deliver a pace. If natural activity is detected, the controller controller should not monitor the heart a refractory period, a.k.a. Atrial Refractory Period (`ARP`), with `ARP` < `Period`, since some heart activity is present after a natural event or pace.

The students had to design a controller that regulates the behavior of the pacemaker for the different modes using Simulink®.

3 Simulink® Models

This section describes the two Simulink® models (`model 1` and `model 2`) of the pacemaker controller designed by the students for the 3MD3 course.

Model 1. Figure 4 presents a portion of the Simulink® `model 1` describing the behavior of the controller in the AAI mode. The model is designed using Simulink® Stateflow, a graphical language that includes state transition diagrams.

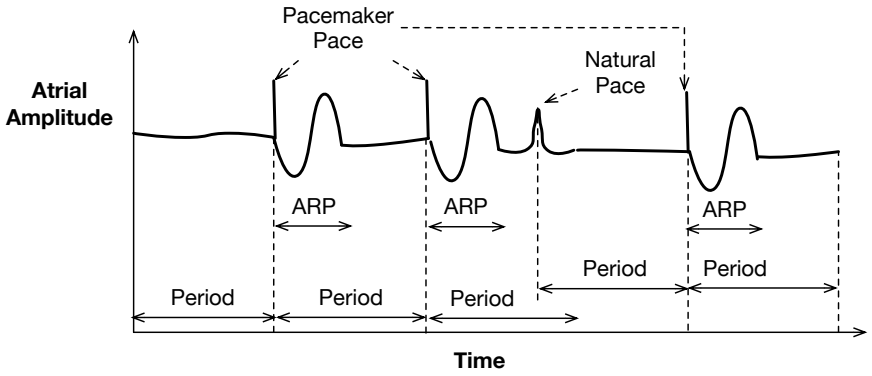


Figure 3: Expected behavior of the controller in the AAI mode.

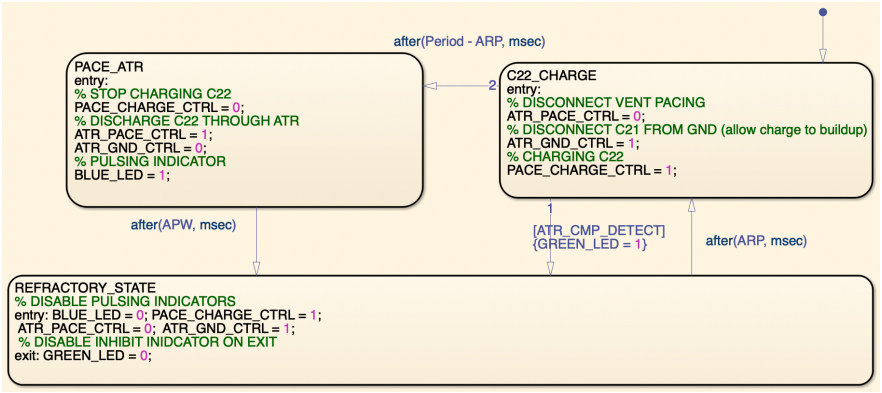


Figure 4: A portion of the Simulink® model 1.

The model has three states: `PACE_ATR`, `C22_CHARGE`, and `REFRACTORY_STATE`. The `C22_CHARGE` state implements the preparation step described in Section 2 while sensing the atrium. The `PACE_ATR` state paces the heart. The `REFRACTORY_STATE` forces the controller to avoid sensing the heart for time `ARP` after a natural pace is detected, or the pacemaker paced the heart.

The controller starts in the `C22_CHARGE`. State `C22_CHARGE` mantains the capacitors `C22` and `C21` respectively charged and discharged while sensing for a natural pace. If a natural pace is detected within time interval `Period`, the controller moves to the `REFRACTORY_STATE`. Otherwise, it moves to the state `PACE_ATR`. State `REFRACTORY_STATE` ensures that the controller spends time `ARP` without sensing the heart. After time `ARP` is passed, the controller returns to the state `C22_CHARGE`. State `PACE_ATR` paces the heart. After pacing, the controller moves to the state `REFRACTORY_STATE`.

Model 2. Figure 5 presents a portion of the Simulink® `model 2` describing the behavior of the controller in the AAI mode.

The model has three states: `AAI_Refactory`, `AAI_Sensing`, and `AAI_Pacing`. The `AAI_Refactory` state implements the preparation step described in Section 2, i.e., it charges the capacitor `C22` and discharges the capacitor `C21`. The `AAI_Pacing` state implements the pacing step described in Section 2, i.e., it discharges the capacitor `C22` and charges the capacitor `C21`. The `AAI_Sensing` state senses whether there is a natural pace within the time interval `Period`.

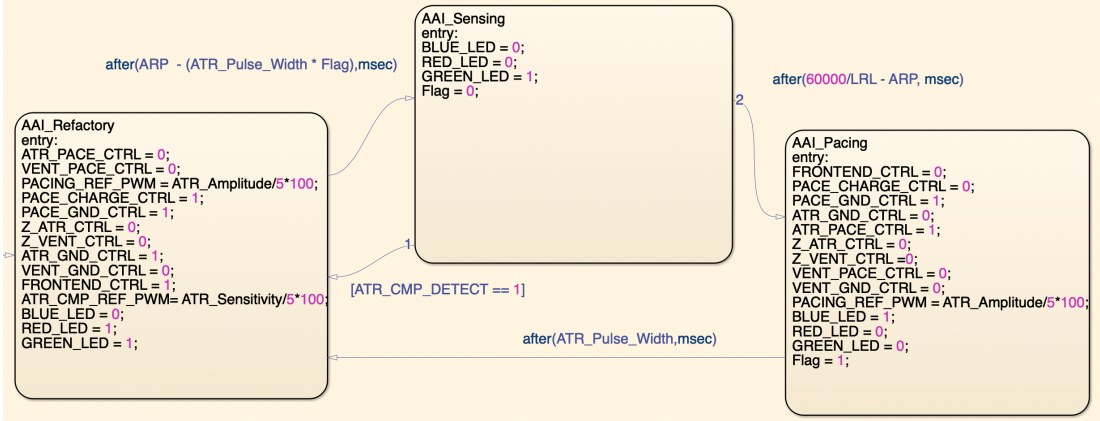


Figure 5: A portion of the Simulink® model 2.

The controller starts from the `AAI_Refactory` state. After charging and discharging the capacitors `C22` and `C21`, it moves to the `AAI_Sensing` state. When the controller is in the `AAI_Sensing` state, if it detects a natural pace within the time interval `Period`, it returns to the `AAI_Refactory` state to ensure the capacitors are properly charged before restarting the sensing. If the controller does not detect any natural pace within the time interval `Period`, it moves to the `AAI_Pacing` state. In the `AAI_Pacing` state, the controller paces the heart and sets 1 as the value for the variable `Flag`. Then, it moves to the `AAI_Refactory` state. The value 1 assigned to the variable `Flag` ensures that the controller spends a time `ARP` in the `AAI_Refactory` state ensuring that the heart is not sensed for this period.

The students had to define a set of requirements of their choice during the controller design and use falsification-based testing to check for requirement violations.

4 Requirements and STL Formalization

This section presents two requirements, respectively for `model 1` and `model 2`, and their formalization in Signal Temporal Logic (STL) [9].

Requirement 1. The STL formalization for the requirement ϕ_1 of `model 1` is as follows.

$$\phi_1 := \mathcal{G}_{[0,10]}(\texttt{PACE\_COUNT} \leq 15) \wedge \mathcal{F}_{[0,10]}(\texttt{PACE\_COUNT} \geq 8)$$

where $\mathcal{G}$ and $\mathcal{F}$ are respectively the globally and eventually STL operators. The requirement specifies that the number (`PACE_COUNT`) of pacing within the time interval $[0, 10]$s shall be (a) lower than 15 across the all interval $[0, 10]$s, and (b) to eventually become greater than 8.

The Pacemaker System Specification document [12] specifies that the lower rate limit (LRL) range is $50 - 90$ Beats Per Minute (BPM). The LRL indicates the desired amount of beats per minute for the pacemaker. Considering 90BPM as maximum value for the BPM and 10s ($1/6$m) as simulation time, the number of beats detected should be lower than 15 paces (i.e., $\lceil 90 * 1/6 \rceil$) within the interval $[0, 10]$s. Considering 50BPM as minimum value for the BPM and 10s ($1/6$m) as simulation time, the number of beats detected should be higher than 8 paces (i.e., $\lfloor 50 * 1/6 \rfloor$) within the interval $[0, 10]$s.

Requirement 2. The STL formalization for the requirement ϕ_2 of `model 2` is as follows.

$$\phi_2 := \mathcal{G}_{[0,10]}(\texttt{VENT\_CMP\_REF\_PWM} \leq 100)$$

The requirement specifies that Pulse Width Modulation (PWM) frequency (`VENT_CMP_REF_PWM`) shall be lower than 100Hz.

The students used falsification-based testing to check for requirement violations.

5 Falsification-based Testing

This section describes the results of the falsification-based testing activity conducted by the students. Specifically, we describe the results obtained by considering a yet-under development version of the Simulink® model for the pacemaker controller. The results are obtained using S-TaLiRo [3], a well-known tool for falsification-based testing of Simulink® models. We ran each experiment 50 times as mandated by the ARCH competition [6] – an international competition among testing tools for continuous and hybrid systems [4] that is held as a part of the international conference on computer safety, reliability, and security (SAFECOMP) [1].

Testing Requirement 1. The model considered for testing this requirement had a problem in the condition that triggers one of the transitions of the Stateflow diagram. Specifically, the condition of the transition connecting the state `CHARGE_CAPACITOR` with the state `PACE_ATR` is set to "`after`(`Period` − 70 ∗ `APW`)" within the failing model instead of "`after`(`Period` − `APW`)".

The students tested requirement ϕ_1 by considering two testing scenarios. In the first scenario (*Scenario 1*), they assumed that the value of the desired lower rate limit (`LRL`) changes across the simulation. Specifically, the value of the variable `Period` (see Figure 5) is equal to $^{1m}/_{\texttt{LRL}}$. For the input generation, the students considered $[50, 90]$ as input range, 5 control points, one each 20s, and the Piecewise Cubic Hermite Interpolating Polynomial (`pchip`) interpolation function [11], since it generates smooth and continuous signals. In the second scenario (*Scenario 2*), they also considered the mode of the pacemaker as input. They considered $[1, 4]$ as input range since the values $1, 2, 3, 4$ represent the different modes of the controller, 5 control points, one each 20s, and the Piecewise Constant (`pconst`) interpolation function, since it generates constant values over consecutive intervals. For both the scenarios, the students considered UR_TaLiRo as uniform random generates samples fairly distributed across the input domain.

Table 1 reports the falsification rate (*FR*), i.e., the number of runs (over 50 runs) for which S-TaLiRo detected a requirement violation. It also reports the mean ($\overline{S}$) and median ($\widetilde{S}$) across the different runs of the number of model simulations required to detect the requirement violation. The results of Table 1 show that, for *Scenario 1* and *Scenario 2*, S-TaLiRo detected a failure-revealing input for respectively 50 and 50 runs (out of 50). When S-TaLiRo found a failure-revealing input for *Scenario 1* and *Scenario 2*, the average number of iterations required to detect the input was respectively 4.2 and 3.0, the median was respectively 1.1 and 1.0.

When the model was fixed (see Table 1), i.e., model `model 1′` was generated, S-TaLiRo still generated failure-revealing test cases indicating the presence of other faults within the model.

Testing Requirement 2. The model considered for testing this requirement had a problem: the value assigned to the output signal `VENT_CMP_REF_PWM` by the state `VVI_Refactory` is 125Hz instead of `VENT_CMP_REF_PWM = VENT_Sensitivity/5*100`

The students tested requirement ϕ_2 by considering two testing scenarios. In the first scenario (*Scenario 1*), the value of the amplitude for the pace of the ventricle (`VENT_Amplitude`) and the mode of the pacemaker (`Mode`) change across the simulation. The students set 15s as simulation time. For the input generation, the students considered $[1, 5]$ as input range, 3 control points, one at the beginning, middle, and end of the simulation, and the `pchip` interpo-

Requirement	Model	Scenario	FR	$\overline{S}$	$\widetilde{S}$
Requirement 1	`model 1`	*Scenario 1*	50	4.2	3.0
	`model 1`	*Scenario 2*	50	1.1	1.0
	`model 1′`	*Scenario 1*	13.0	136.6	100.0
	`model 1′`	*Scenario 2*	50.0	1.02	1.0
Requirement 2	`model 2`	*Scenario 1*	50	5.3	3.0
	`model 2`	*Scenario 2*	50	4.2	3.0
	`model 2′`	*Scenario 1*	0.0	-	-
	`model 2′`	*Scenario 2*	0.0	-	-

FR: falsification rate wrt. number of 50 trials,
$\overline{S}$ and $\widetilde{S}$: mean resp. median (rounded down) number of simulations over successful trials.

Table 1: Results of the testing activity of the pacemaker benchmark.

lation function [11] for the input signal `VENT_Amplitude`, and 3 control points and the `pconst` interpolation function for the `Mode` input signal. In the second scenario (*Scenario 2*), they also assumed that the `VENT_Pulse_Width` changes across the simulation. For this signal, they considered $[0, 2]$ as input range, 3 control points, and the `pchip` interpolation function.

The results of Table 1 show that, for both the scenarios, S-TaLiRo detected a failure-revealing input for 50 runs (out of 50). The average number of iterations required to detect the failure-revealing input was respectively 5.3 and 3.0, the median was respectively 4.2 and 3.0.

When the model was fixed (see Table 1), i.e., model `model 2′` was generated, S-TaLiRo could not generate any failure-revealing test case.

6 Discussion

The pacemaker benchmark contains a representative model from the medical domain. First, the pacemaker benchmark was defined by consulting the pacemaker system specification [12] provided by Boston Scientific [5]. Second, the Simulink® models of the controller of the pacemaker were deployed and tested on a FRDM-K64F microcontroller [7]. A video of the behavior of one of the controllers when deployed on a FRDM-K64F microcontroller is available online [8].

The pacemaker benchmark can be added to the models of the ARCH competition [6]. The ARCH competition considers contains different benchmarks. However, it does not contain any benchmark from the medical domain. Adding the pacemaker benchmark to the ones considered by the ARCH competition will overcome this limitation. For example, the *Scenario 1* of the `model 1′` can be a valuable example for comparing existing tools since (a) S-TaLiRo found a failure-revealing input in a limited number of runs (13.0 out of 50 runs), and (b) S-TaLiRo required a considerable number of iterations (on average 136.6).

7 Conclusion

This paper proposes a benchmark containing two Simulink® models for a simple pacemaker controller. Each model comes with a requirement formalized in STL. The models and requirements can be input in existing falsification-based testing tools, such as S-TaLiRo [3] and Aristeo [10]. We presented and discussed the results obtained with S-TaLiRo [3].

We argued that the proposed benchmark is representative since it was defined by considering the pacemaker system specification [12] provided by Boston Scientific [5] and the controllers were deployed and tested on the FRDM-K64F microcontroller [7]. We also suggested adding the pacemaker benchmark to the ARCH competition.

In future work, we plan to evaluate alternative STL formalizations for our requirements (e.g., by using external variables to count the heartbeats within a sliding window), to split Requirement 1 into two alternative requirements (one for each subformula of the conjunction), and to assess the impact of these alternative STL formalizations on our results. Finally, we will evaluate the impact of the number of control points, interpolation function, and simulation time on the falsification results.

Acknowledgments

We thank Alan Wassyng, Guy Meyer, Michael Kehinde for designing and improving the pacemaker assignment of the 3MD3 course over the years.

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference numbers RGPIN-2022-04622, DGECR-2022-00406].

References

[1] *Computer Safety, Reliability, and Security (SAFECOMP)*. Springer, 2021.

[2] Simulink. https://www.mathworks.com/products/simulink.html, 06 2021.

[3] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257. Springer, 2011.

[4] International Competition on Verifying Continuous and Hybrid Systems. https://cps-vo.org/group/ARCH/FriendlyCompetition, 04 2022 [Online].

[5] Boston Scientific. https://www.bostonscientific.com/en-US/Home.html, 04 2022 [Online].

[6] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Aniruddh Chandratre, Alexandre Donzé, Georgios Fainekos, Goran Frehse, Khouloud Gaaloul, Jun Inoue, Tanmay Khandait, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Masaki Waga, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. ARCH-COMP category report: Falsification with validation of results. In *Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 80, pages 133–152. EasyChair, 2021.

[7] FRDM-K64F. https://www.nxp.com/design/development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F, 04 2022 [Online].

[8] Replication Package. https://github.com/3MD3/3MD3.git, 04 2022 [Online].

[9] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[10] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification. In *International Conference on Software Engineering*, pages 372–384. IEEE/ACM, 2020.

[11] Piecewise Cubic Hermite Interpolating Polynomial (PCHIP). https://it.mathworks.com/help/matlab/ref/pchip.html, 04 2022 [Online].

[12] Boston Scientific. Pacemaker system specification. *Boston Scientific*, 2007.