Towards Certifiable Software for Medical Devices: The Pacemaker Case Study Revisited

Michaela Huhn Sara Bessling
Department of Informatics, Clausthal University of Technology
38678 Clausthal-Zellerfeld, Germany
email{Michaela.Huhn|Sara.Bessling}@tu-clausthal.de

Design and verification of pacemaker software - as an instance of a highly dependable medical device - has been investigated in numerous works tackling various safety requirements with different formal methods. However, in order to certify a product, a conclusive argument has to be provided that seamless and concerted safety activities starting from the hazard analysis towards the verification of the derived safety requirements yield a dependable product.

We present an approach towards the development of certifiable medical device software using SCADE Suite for the pacemaker case study. For safety analysis we use *Deductive Cause Consequence Analysis* (DCCA) as an enhanced, systematic technique to identify potential hazards and verify the derived the safety requirements. Formal verification is split into a part done in the SCADE Suite and the real-time behavior which is proven using UPPAAL.

1 Introduction

Within the prospering markets for health care, the area of medical devices is thriving as well. Numerous new application areas, e.g. for living assistance or home-based medical support, have been developed based on the emerging possibilities of software-controlled devices. Dependability was an issue for medical devices always, but only 2006 a safety standard, i.e. IEC 62304 [7] that regulates the software life cycles activities for medical devices, was agreed on. IEC 62304 names quality goals as well as core processes and development activities that are well-suited for dependable software. But in difference to other domains like IEC 61508 [2], the standard's recommendations are not underpinned by concrete techniques that are considered appropriate for a certain software integrity level. Thus a commonly agreed or at least scientifically justified line of methods for safety development of medical devices is still missing.

We demonstrate a model-based, formally founded approach to the development of safety-critical software on the case of a cardiac pacemaker. Our approach, as we present it here, provides coordinated hazard analysis, model-based design with automated code generation and formal verification by model checking. We use DCCA [9] as a systematic and formal method to identify hazards and derive safety requirements. These are modeled as observer nodes and reused for verification later on. We decided for SCADE Suite by Esterel Technologies [4] as development framework, since SCADE Suite has been formally qualified as an adequate tool for developing software for safety-critical systems compliant to safety standards (see [4]). Tool qualification is a key argument in a safety case that has to be provided for certification, because in the safety case appropriateness of methods has to be proven.

In a first attempt the design was done fully in SCADE, but efficiency arguments from both, design and verification, turned the decision towards an external handling of timers and events. Consequently, we have to verify the real-time behavior separately from the control logic. For this we extend the time abstraction we proposed in [3] for a case study on a railway level crossing modeled in SCADE as well.

The contribution of this paper is twofold: (1) The combined verification of the safety requirements using SCADE Design Verifier and UPPAAL is of interest in itself. (2) We give a showcase for a concerted line of methods for safety development on which a safety argument can be built as backbone for certification.

2 Background

2.1 The SCADE Tool Suite

The acronym SCADE stands for Safety-Critical Application Development Environment. The main objectives of the SCADE Suite are (1) to support systematic, model-based development of correct software based on formal methods and (2) to cover the whole development process [4]. The language *Scade* underlying the tool is data-flow oriented. Its formal semantics is based on a synchronous model of computation, i.e. cyclic execution of the model.

The SCADE Suite is an integrated development environment that covers many development activities of a typical process for safety-critical software: modeling, formal verification using the SAT-based SCADE Design Verifier [1], certified automatic code generation producing readable and traceable C-code, requirements tracing down to model elements and code, simulation and testing on the code level and coverage metrics for the test cases with the Model Test Coverage module.

2.2 Deductive Cause Consequence Analysis

A major goal in safety analysis is to determine how faults modes at the component level causally relate to system hazards. Among the various formally founded techniques proposed for this task we have selected *Deductive Cause Consequence Analysis (DCCA)* by Ortmeier et al. [9, 5], because DCCA does not only formalize techniques like FTA (*Fault Tree Analysis*) and an FMEA (*Failure Mode and Effect Analysis*) [6], which are well-established and recommended by the standards. In addition, the identified fault modes and hazards can be reused in safety assurance to formally verify that sufficient measures have been taken to prevent the identified hazards. In DCCA, components faults are modeled as simple fault automata that extend the normal behavior of the component. Hazards are specified as observer nodes that read signals from the control logic and evaluate them according to the *negation* of the hazard predicate. Then the verification process can be performed in order to iteratively determine the so-called *minimal critical sets*, i.e. subsets of faults that may lead to the hazard - in case that they occur in a certain order (for details and formalization see [5, 3]).

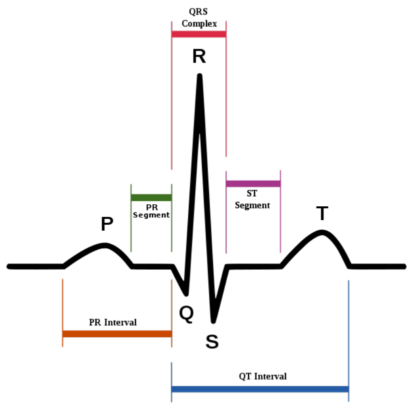


Figure 1: Sinus rhythm of a human heart (sane adult), source: Wikipedia

2.3 Related Work

Since PACEMAKER Formal Methods Challenge was set up by publishing Boston Scientific's Specification of an industrially produced pacemaker [10], pacemakers were investigated intensively within the formal methods community. For brevity, we only refer to two of them: Jee, Lee, and Sokolsky worked on assurance cases of the pacemaker software [8]. The authors focused on the basic VVI mode of a pacemaker, and employed UPPAAL for both, design and verification. Moreover, they implemented their own code generation from UPPAAL to C-code to generate an executable from their UPPAAL model. In [11] the authors used timed CSP to verify some of the timing constraints for different pacemaker modes. However, only the specification level is considered in this work.

3 The Pacemaker

3.1 The Human Heart

From a bio-mechanical point of view, the human heart is the pump of the circulatory system. It consists of two atria and two ventricles. The contraction of the heart is initiated at the so-called *sinoatrial node (SA node)*, an area of self-excitable cells within the right atrium known as *P wave*. The electrical impulses spread through the atria and ventricles with a dedicated timing characteristics (see the *electrocardiogram (EKG)* and shown in Figure 1), thereby causing the contraction of the chambers.

Normally, the SA node generates electrical impulses with a frequency of 60-100 beats per minute. A too low or sporadically missing pulse generation is called *bradycardia*. In order to support the natural pulse generation notably for bradycardia, artificial cardiac pacemakers are implanted nowadays. Artificial pacemakers have to respect the timing characteristics of the sinus rhythms as it is critical. First and foremost, pulses must not be generated within the refractory intervals after depolarization, as this may cause life-threatening cardiac fibrillation.

3.2 Informal Specification of a Pacemaker

In this paper we mainly describe a modern, atrium-controlled DDD pacemaker [10], although we analyzed a whole family of pacemakers. Beginning with the least complex A00/V00 and D00 pacemakers which stimulate the heart periodically with a fixed time interval, over AAI/VVI pacemakers which sense the chamber's signals and stimulate one of the chamber only when a signal is missing, till the most complex pacemaker DDD monitoring and stimulating both chambers.

DDD means dual pacing, dual sensing, and dual response mode, see the NBG code for details of the configuration. A DDD pacemaker senses both right chambers and can also stimulate them both, but only if no natural pulses are detected. The DDD pacemaker basically uses two timers to monitor time intervals: The *base* interval is the period between two subsequent P waves, natural or artificial, of the atrium. The *AV interval* is the time between a stimulation of the atrium and the consecutive ventricle pulse, the *QRS complex*. If the base interval expires without sensing a natural P wave in the atrium, an artificial impulse is generated in the atrium. Then the ventricle is monitored and only in case of no natural pace within the AV interval, an artificial ventricle pulse is generated. Both timers are reset in case an appropriate natural impulse is sensed. In addition, the base interval is reset if a ventricular extra-systole occurs. Then the base interval is restarted without starting the AV interval timer again. In case the SA node generates a natural pace, the AV interval is adapted (so-called AV hysteresis) in the next base period. For now, we consider the base interval and thereby the pace frequency to be fixed.

4 The Safety Process

As prescribed in the safety standards [7, 2], the system development is complemented by a safety analysis that identifies hazards and traces them back to potential failures. From the identified hazards system safety requirements are derived to eliminate failures or mitigate their effects. In consistency with the architectural decomposition of the system into components, the safety requirements are split into sub-requirements that are assigned to individual components. The safety analysis and the decomposition of components and safety requirements are iterated until basic components are derived that can be realized and for which evidence can be provided that they fulfill their safety requirements.

4.1 Safety Analysis

The principal architecture of a pacemaker is depicted in Figure 2. In the safety analysis we concentrate on those hazards and the induced safety requirements that refer to the *functional level* of the software control of the pacemaker whereas the mechanics, the electrics, and the deployment are analyzed no further. We performed an FTA and an FMEA [6] and formalized it according to the DCCA approach [5]. The resulting safety requirements for the functional level of the software control are as follows:

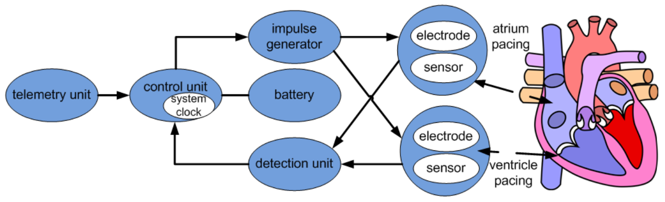


Figure 2: Principal architecture of a cardiac pacemaker

Timed Interrupt: As a pacemaker is limited by its battery and its replacement requires an operation, energy consumption of all components has to be kept as low as possible. *Refractory periods*: Within the refractory periods after the atrium (ARP) and ventricle pace (VRP), detection and impulse generation have to pause in that chamber in order to guarantee that neither an artificial pulse is sensed nor disturbances after depolarization are misinterpreted. *Time intervals BI, AVI + AVH:* The timing constraints as the base interval, the AV interval with the AV hysteresis and their sequencing are respected within specified tolerances. *Pacing:* An artificial atrium pace is triggered if the base interval expires without sensing a natural P wave. An artificial ventricle impulse is generated if the AV timer has been started and expires without sensing a natural pace there. If detection is active it suspends the impulse generation for that chamber and vice versa.

4.2 Safety Design

When using SCADE suite, the C-code generated from the SCADE design model is embedded into a wrapper that is usually periodically executed. Thereby it is ensured that the model reacts synchronously. Within each execution cycle all input signals are read and all output signals are written. For the pacemaker, the control logic should be executed only if a control state or one of the output signals is about to change. In those cycles in which both refractory periods overlap and the software control only waits for the ARP timer to expire, model execution may pause in order to save energy. Thus in order to meet the efficiency request *Timed interrupt* we decided to handle timers and events (from the detection unit) outside the SCADE model and call the inner part of the control modeled in SCADE only to react on timeouts and event occurrences. As a consequence the verification task has to be decomposed: Since the real-time behavior know is realized by both, the SCADE model and the wrapper, it can be proven correct only partially on the SCADE level. The overall reactivity has to be verified separately. For this task we will employ UPPAAL [12], a model checker based on timed automata that is capable to deal with real-time constraints. The safety requirements referring to the inner control logic are proven using SCADE Design Verifier. The inner control is modeled as a state machine on the top level (see Figure 4).

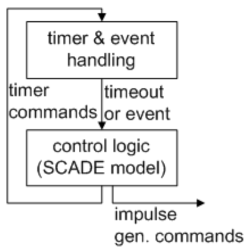


Figure 3: Event handling

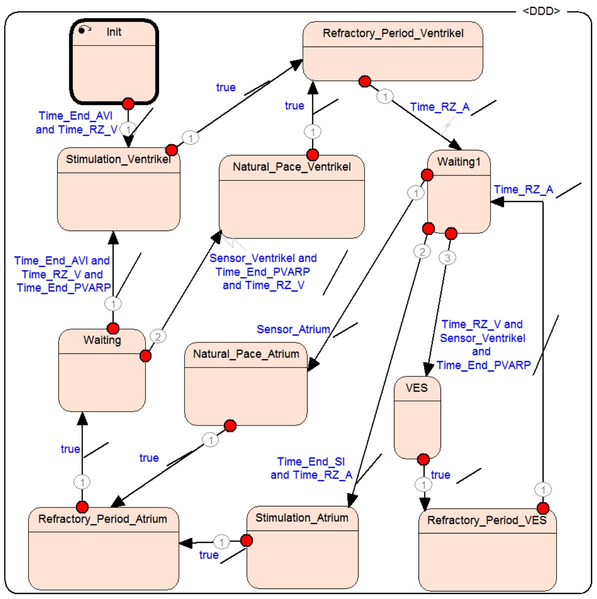
In the first glance, the alternative, namely doing the control fully within SCADE seems to have the advantage of a simple and seamless design *and* verification methodology. However, a pure SCADE solution suffers not only from inefficient execution but verifying the timing behavior with SCADE Design Verifier is seriously affected by complexity problems: In a pure SCADE model, the system clock is referred as an external signal that has to be compared with the timeout value (integer) in each execution cycle. From the design perspective this corresponds to polling. When the state space is explored for verification, all intermediate states are traversed which corresponds to successively increasing the timer cycle by cycle. SCADE Design Verifier provides SAT-based model checking performed on the transition graph representing the formal semantics of the design model. The SAT-Solver is enhanced by a number of built-in abstraction techniques, in particular integer linear arithmetics [1]. We had expected that to be an effective tool, in particular for handling the comparisons with timing constants. However, it was not able to cope with this way modeling the timers.

We have shown in [3] that time abstraction is very promising in order to cope with this kind of state explosion caused by a real-time clock.

4.3 Safety Assurance

4.3.1 Verifying the Timing Constraints

The timer and event handling can be modeled in UPPAAL in a straightforward manner: Each relevant interval is monitored by a clock, events like the sensing or stimulating a pace are modeled as communication and the statuses of these timers are represented in variables. The possible statuses of a timer are *inactive, init, active*, and *timeout*, and they can be considered as the input signals for the control logic modeled in SCADE. The simplest way to complete the UPPAAL model is to construct a timed automata for the inner control logic as well by using the model transformation we proposed in [3]. The real-time constraints are expressed using the UPPAAL query language. At this level the real-time constraints are specified stating that within certain real-valued intervals certain timer statuses are set, or certain events must or must not occur. E.g. whenever a ventricle pace has been sensed or stimulated the timer monitoring the ventricle refractory period is set active for a time constant VRP.



<DDD>
Init
Refractory_Period_Ventrikel
Time_End_AVI and Time_RZ_V
true
true
Time_RZ_A
Stimulation_Ventrikel
Natural_Pace_Ventrikel
Waiting1
Time_RZ_A
Time_End_AVI and Time_RZ_V and Time_End_PVARP
Sensor_Ventrikel and Time_End_PVARP and Time_RZ_V
Sensor_Atrium
Time_RZ_V and Sensor_Ventrikel and Time_End_PVARP
Waiting
Natural_Pace_Atrium
VES
true
true
Time_End_SI and Time_RZ_A
true
Refractory_Period_Atrium
Stimulation_Atrium
Refractory_Period_VES
true

Figure 4: DDD pacemaker: Inner control logic

4.3.2 Verifying the Control Logic

We verified several safety requirements referring to the control logic that we derived in the safety analysis. During the verification process the SCADE Design Verifier tries to find an input configuration which changes the output of the proof obligation from true to false. If the output stays true for each possible input configuration, the output is considered as valid. First pausing of the detection and impulse generation

	Natural pace	**Natural pace atrium**	**Refractory period**	**No VES**	**Hysteresis**	**One pace**
VVI	x (0 s)		x (0 s)			x (3 s)
AAI	x (0 s)		x (0 s)			x (3 s)
DDD	x (1 s)	x (0 s)	x (0 s)	x (1 s)	x (0 s)	x (1 s)

Table 1: The correlation of constraints and pacemaker modes, the runtime is shown in brackets

during refractory periods is verified. For this we argue that it shall always be true that while a timer for a refractory period is running, neither the sensor is active nor a stimulation is triggered. We verified the pacing requirement for each chamber separately. Moreover, the pacing requirement is divided into two parts: The first sub-requirement concerns the case of a natural pace, the second one the artificial stimulation. For the first part we argue similar to the previous constraint that it is not possible that the statuses of the timers allow pace sensing and a pace is sensed as well as a stimulation takes place. We call the constraints *(*natural pace BI) for the base interval and *natural pace AVI* for the AV interval. Furthermore we have a constraint called *refractory period* in which we verify that during the refractory periods no sensing or stimulation takes place. For the second part we argue that if no natural pace is sensed during the corresponding base or AV interval, exactly one artificial pace takes place. This constraint is called *one pace*. To verify this requirement we created an operator in which the natural and artificial paces during one interval are counted. For the timing constraint AVH we verified that after sensing a natural ventricle pace the next AV interval will be prolonged. This is done by saving the ventricle pace and waiting for the beginning of the AV interval. At that point we control the length of the AV interval. Furthermore we verified the correct handling of a ventricular extrasystole (VES). This constraint is called *VES*. In the corresponding proof operator we determined that it shall not be possible that a VES and an atrial pace (natural or artificial) take place together in one base interval. For this we memorize in an operator the paces that occurred within an base interval. As result we received for all six proof obligations to be valid. The runtimes for each verification are shown in table 1. Due to the particular characteristics of the different pacemaker variants, not every constraint is required and consequently guaranteed for every pacemaker. In table 1 we oppose the constraints for the AAI/VVI pacemaker with the ones for the DDD pacemaker.

If we fully integrated the timers into the pacemaker logic, the verification process did not terminate within a reasonable time of two days. We set this boundary out of our experience with the SCADE Design Verifier in combination with the model complexity. If we wait longer for results we experienced no results at all because of memory overflow or buffer overflow. This negative result also justifies the architectural design as depicted in Figure 3.

5 Conclusion

We sketched how to systematically develop a safety-critical embedded system using formal methods. We employed the SCADE suite for modeling and code generation for the inner control logic, as SCADE is qualified to the most relevant safety standards like IEC 61508 and RCTA DO 178-B. Timers and events were handled in an outer control loop. This decomposition was motivated by an efficiency requirement. For verification purposes it can be understood as a time abstraction which turned out to be a useful also for proving the real-time behavior correct. The approach presented here extends the ideas of time abstraction we presented in [3]. In that work, pure SCADE models were embedded into a standard cyclic wrapper and the time abstraction was performed on the SCADE model. Here timers and simple event handling are transferred to the wrapper on which is an abstraction to UPPAAL is applied.

In the full version of this paper, the safety constraints referring to real-time behavior as well as to the control logic will be detailed. Moreover, an alternative proof strategy based on assume-guarantee style will be explored.

References

[1] Parosh Aziz Abdulla, Johann Deneux, Gunnar Stålmarck, Herman Ågren & Ove Åkerlund (2004): *Designing Safe, Reliable Systems Using Scade*. In Tiziana Margaria & Bernhard Steffen, editors: *ISoLA*, *LNCS* 4313, Springer, pp. 115–129. Available at `http://dx.doi.org/10.1007/11925040_8`.

[2] Intern. Electrotechnical Commission (2010): *IEC 61508-3:2010: Functional safety of electrical/electronic/programmable electronic safety-related systems Part 3: Software requirements*.

[3] Ilays Daskaya, Michaela Huhn & Stefan Milius (2011): *Formal Safety Analysis in Industrial Practice*. In Gwenn Salaün & Bernhard Schätz, editors: *16th Intern. Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, LNCS 6959, Springer, pp. 68–84.

[4] Esterel Technologies (2009): *SCADE Suite KCG 6.1: Safety Case Report of KCG 6.1.2*.

[5] Matthias Güdemann, Frank Ortmeier & Wolfgang Reif (2007): *Using deductive cause-consequence analysis (DCCA) with SCADE*. In: *Proc. 26th Intern. Conference on Computer Safety, Reliability and Security (SAFECOMP)*, *Lecture Notes Comput. Sci.* 4680, Springer, pp. 465–478.

[6] International Electrotechnical Commission (2006): *IEC 60812: Analysis Techniques for System Reliability*.

[7] International Electrotechnical Commission (2006): *IEC62304: Medical device software - Software life-cycle processes*.

[8] Eunkyoung Jee, Insup Lee & Oleg Sokolsky (2010): *Assurance Cases in Model-Driven Development of the Pacemaker Software*. In Tiziana Margaria & Bernhard Steffen, editors: *4th Intern. Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, *LNCS* 6416, Springer, pp. 343–356.

[9] Frank Ortmeier, Wolfgang Reif & Gerhard Schellhorn (2006): *Deductive Cause Consequence Analysis (DCCA)*. In: *Proc. IFAC World Congress*, Elsevier, Amsterdam.

[10] Boston Scientific (2007): *PACEMAKER System Specification*.

[11] Luu A. Tuan, Man C. Zheng & Quan T. Tho (2010): *Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker*. In: *4th Conf. on Secure Software Integration and Reliability Improvement*, IEEE, pp. 23–32.

[12] (2009): *UPPAAL 4.0: Small Tutorial*. `http://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf`. November 16, 2009.