MASTER

Formal time-based modeling and verification of industrial machines

Adyanthaya, S.

Award date:
2011

Link to publication
Formal Time-based Modeling and Verification of Industrial Machines

Shreya Adyanthaya
0754541

August 19, 2011

Master Thesis
Eindhoven University of Technology
Department of Mathematics and Computer Science
Chair of Software Engineering and Technology

Supervisor at TU/e:
dr. Suzana Andova
Assistant Professor
s.andova@tue.nl

Supervisor at MIT:
prof.dr. Radhika M. Pai
Head of Department
radhika.pai@manipal.edu

Graduation Tutor:
dr.ir. Johan Jacobs

Graduation Tutor:
dr. Istvan Nagy
Abstract

Industrial embedded systems are hardware systems that have associated software to operate them and which perform specific functionality. Such systems are often highly concurrent with several parallel components operating together. These components have synchronizations and dependencies between them that make such systems highly complex. It is often very beneficial to use formal approaches to analyze their complex behavior, prior to the actual programming of their system software. These formal approaches are used to design models of completely new systems as well as to evaluate the design of upgrades to the systems. The correctness of these models becomes very crucial since they then form input for the design of the actual software that controls the operations of the system hardware. Simulation is one approach that allows the analysis of certain behavior aspects of the system. However, it is also essential to ensure that the system behaves accurately under all possible circumstances. Hence, there arises a need for model checking which allows verification of required properties on the system for all its dynamic behaviors.

Embedded systems with hard real time constraints are very time-specific and require the verification of time-specific logistic rules. In such cases, model checking approaches that also allow time based verification need to be considered. The main purpose of this work is to perform time based model checking on such an industrial system. The various stages of this work are briefly summarized below:

- Understanding the dynamics of the industrial system
- Understanding the time specification that is currently used for modeling the system
- Investigating which language and model checker is most suitable and can handle a real industrial case study for its complete dynamic behaviors.
- Modeling and verification of the system. There are several properties that need to be analyzed on the models such as:
  - Detecting or proving the absence of deadlocks in the system.
– Proving various time-based logistic rules. These properties mainly consider the timing behavior of the system.

• Diagnosis of verification results and suggesting approaches to overcome issues, if any.

Accurate time-based verification requires the translation of the current design models of the system to formal semantics of existing modeling languages and analysis of the entire state space of the system by means of state-of-the-art model checkers. Several model checkers are suitable for this kind of verification such as Uppaal, mCRL2 and other temporal logic model checkers. These model checkers can be used to model highly concurrent systems and perform verification of required properties.
Acknowledgements

This thesis is written as a part of the master project that I have performed under the chair of Software Engineering and Technology at the Eindhoven University of Technology, The Netherlands, in close cooperation with ASML, Veldhoven, The Netherlands.

Upon completion of my master thesis, I would like to extend my heartfelt gratitude to everyone who has contributed to the successful completion of this work. Firstly, I would like to thank my supervisors, dr. Suzana Andova, dr.ir. Johan Jacobs, and dr. Istvan Nagy without whose support and guidance this work would not have been possible. My university supervisor, dr. Suzana Andova, gave me all the freedom to explore the various possibilities in my work during our discussions along with encouraging me to come up with innovative ways of solving problems. I learnt a lot during our interactions and her guidance on both, a professional and a personal level, has been incomparable. I would like to thank my graduation tutor at ASML, dr.ir. Johan Jacobs, for giving me such an interesting assignment and teaching me with utmost patience, all the aspects of the ASML machines. Working under him has been not only challenging but also a fun experience altogether. I must thank my ASML supervisor, dr. Istvan Nagy, for choosing me for this project and guiding me throughout with all the various possibilities that I could explore in my work, along with enlightening me with several experiences from his past works. I also wish to thank my supervisors for patiently reviewing this thesis several times and for their valuable comments.

I extend my gratitude to prof.dr. Mark van den Brand, the program director for the Manipal-TU/e dual degree program, for being a part of my assessment committee in my project and most importantly for selecting me, along with prof.dr. Manohara M. Pai and prof.dr. Jos Baeten, to do my masters in this prestigious institute. I owe my extreme gratitude to them for giving me this opportunity. I thank my supervisor at Manipal Institute of Technology, dr. Radhika M. Pai, for being my project supervisor from Manipal and for her valuable evaluation. I express my gratitude to dr. Mohammad Mousavi for being a part of my assessment committee and for his valuable suggestions on my thesis work. I also wish to express my sincere gratitude to prof. Jan Friso Groote for helping me in making my design decisions with his knowledge and incomparable experience. I also thank dr.ir. Ramon Schifflers (ASML), Sven Weber (ASML), dr.
ir. J.P.M Voeten (ESI), and dr.ir. Bart Theelen (ESI), for their useful guidance during the course of my project. I wish to thank my friends Ajith Kumar and Alok Lele for proofreading my thesis, Muhammad Atif for sharing his knowledge in Uppaal, and all my friends from Manipal University for standing by me at all times. Last but not the least, I wish to thank my family for their undying support and affection which has been my greatest strength at all times.
Model Checking

Model checking \([1]\) is a technique for verifying finite state concurrent systems such as sequential circuit designs and communication protocols. It is the technique of verifying whether a given model of a system meets certain requirements denoted in the form of property specifications. Some such properties are the absence of deadlocks, livelock and reachability properties. It has a number of advantages over traditional approaches that are based on simulation, testing, and deductive reasoning. The main advantage is that while these approaches take into consideration specific sub-behaviors of the system, model checking considers all the possible dynamic behaviors of the system. In other words, all possible states of execution for the system’s model are considered and checked. Hence, if the system is fault free after model checking, it is fault free under all possible circumstances with respect to the model and the properties verified. If the model contains an error, model checking produces a counterexample that can be used to pinpoint the source of the error.

The main challenge in model checking is dealing with the state space explosion problem. Model checking is automatic and usually quite fast. But, as the model of the system becomes larger it results in a much larger number of possible system states. This problem occurs in systems with many components that can interact with each other or systems with data structures that can assume many different values. For these systems, the number of global states can be enormous and as such different approaches need to be used to reduce the state space and perform verification appropriately.

1.1 Formal Modeling Approaches

There are many model checkers/toolsets available today which adopt different concepts toward verification. Since the machines considered and the properties to be verified are
time specific, temporal logic model checkers are most suitable to perform model checking on these machines. Temporal logic is a system of rules and symbolism for representing and reasoning about propositions specified in terms of time. Using temporal logic it is possible to reason about properties stating that an event must always or eventually happen in time, or even properties stating that an event happens following another after a given period of time. Using such constructs it is possible to represent time specific properties crucial for real time systems.

There are several model checkers that can be used to perform this kind of verification. **Uppaal** [2] is the model checker for real time systems that uses the concept of timed automata for modeling purposes and a variant of Computation Tree Logic (CTL) for its specifications. **mCRL2** [3] is a model checker based on a specification language for describing concurrent discrete event systems and which uses mu-calculus [4] for its specifications. **SMV** [5] is the first model checker to use Binary Decision Diagrams (BDDs), which are data structures used to represent boolean functions. **Spin** [6] is used to perform verification on distributed systems and uses Linear Temporal Logic (LTL) to specify its properties. **HyTech** [7] is designed for reasoning about temporal requirements in hybrid systems. Since the system considered is highly concurrent with several processes running in parallel, **Uppaal** and **mCRL2** are suitable for its modeling. **Uppaal** has the advantage of an internal notion of time that can be more efficiently used to represent the time specific behavior of the industrial system. Verification of the system requires the examination of various multiple time models as follows:

- **No time**: This is a model of the system that does not consider the time taken by different actions and mainly considers the different possible sequences of interactions between the components. Verification on a system with no time ensures that all possible scenarios are considered and hence a system which satisfies a property with no time always satisfies the property.

- **Deterministic time**: In this model, specific time durations are assigned to all actions and the system is analyzed for this timing behavior. With deterministic time only a particular time sequence or behavior is considered. If a property is violated in the untimed version of the system, the system is not necessarily incorrect. This is because the property might be satisfied in the deterministic model of the system for all realistic time sequences which can occur in the actual machine.

- **Non-deterministic time**: This model allows actions to occur non deterministically within time intervals. This allows verification to be performed on a system with variable timings within given realistic ranges. Since the industrial systems have timing involved in them and the time duration of activities can vary, the non-deterministic models with time give the most realistic models of the industrial machines. However, since the timing variations are within very small limits, the deterministic models can be efficiently used to analyze the behavior of the system.
As part of the design decision process, a study was performed using both Uppaal and mCRL2, in order to understand the representation of time in these two model checkers and making a choice on the most suitable approach for the verification of the system. mCRL2 is more expressive and can handle very large state spaces. It also allows a visual representation of the state space which can be reduced and viewed using various abstraction techniques. However, mCRL2 does not have an explicit notion of time and this had to be modeled indirectly using its constructs. As a result Uppaal, due to its concept of timed automata, was more appropriate for the modeling of the time-specific industrial systems. Chapter 2 gives the short initial study that was performed using Uppaal before choosing it to perform the complete system modeling.
Chapter 2

Time-based Model of the Tiny Two-way Buffer System

In this chapter, a simple system which has majority of the concepts of the industrial system is presented. This system is modeled and verified in Uppaal using its explicit notion of time. Uppaal has the concept of timed automata where internal clocks, which tick continuously, give the notion of real time.

2.1 Tiny Two-way Buffer System

This is a simple system that replicates, on a small scale, most aspects of the industrial system. It includes parallel processes which synchronize to exchange data with each other along with executing certain delays between these synchronizations. On a much larger scale, a major part of the behavior of the actual system model consists of this behavior due to which this system can be used to perform the initial study of various aspects of the system modeling. In order to understand the various timing aspects that need to be considered with the industrial system, we have taken different versions of the tiny two way buffer system, namely, the system with no time, deterministic time, and non deterministic time.

The system consists of two generators, two buffers, and two exits as shown in Figure 2.1. The first generator G1 repeatedly produces packets. This packet is received over channel $a$ by buffer M1 which sends it over channel $c_2$. The packet is received over this channel by a second buffer M2 which again sends it over channel $e$ to be received by the exit E2. Similarly, the second generator G2 repeatedly generates packets and sends them over channel $b$. This is received by buffer M2 which sends it over channel $c_1$ to buffer M1. M1 again forwards the packet over channel $d$ to exit E1. This system, obviously, has a
deadlock which occurs when both G1 and G2 produce packets at the same time instant. In that case, the packet is received by the buffers M1 and M2 respectively at the same time. As a result, both M1 and M2 wait to send packets to each other and the system deadlocks.

Using the variables A, B and C for delays, the system in Figure 2.1 denotes a general version of the tiny two way buffer system for no time, deterministic time and non-deterministic time. By assigning different values to these variables we get different timing aspects of the system as explained below and summarized in the Table 2.1.

No Time
The system with no time has no time delays in it. Hence, the delay variables are all null in this system as shown in Table 2.1. This system has a deadlock when both generators produce a packet at the same time which are received by M1 and M2. Both the buffers wait for each other to be free so they can forward their packets and the system deadlocks.

Deterministic Time
In this case, some deterministic timing aspects have been added to the system. The generator, G1, produces a packet every A seconds and the generator G2 produces packets every C seconds. Also, after receiving a packet the intermediate buffers M1 and M2 delay for B seconds each before forwarding the packets to the respective exits. This process repeats over time. One possibility is that the variable A is delay 10, B is delay 1 and C is delay 2 as given in Table 2.1. It can be observed easily that there exists a deadlock in the system at time 10 seconds when both generators produce a packet at the same time.

Non-deterministic Time
In this case, a non-determinism has been introduced in the delay of generator G1. As given in Table 2.1 it can produce a packet after a time delay between 9-12. Hence, variable A is delay 9 | delay 10 | delay 11 | delay 12 which means that the system can choose to delay between 9 to 12 seconds. This system also contains a deadlock since no matter when the generator G1 produces a packet, the same or the next second G2 produces a packet and the buffers deadlock.

Figure 2.1: Tiny two-way buffer system
2.2 Uppaal Model without Time

For the tiny two way buffers system with no time, each component of the system has been denoted by a separate automaton in the model. The location marked with red indicates where the control is at the moment. Two global arrays packet and packet1 of the size of the number of packets have been maintained with the initial value for each packet set to 1 indicating that the packet is available.

- G1 and G2 have both been modeled as an automaton with one location and a self loop transition as shown in Figure 2.2 and 2.3. In G1, when the transition is taken, the next value of integer \( n \) (denoting the packet number) is sent over the channel \( a \) by the send operation, \( a[n]! \). As a packet is sent, the corresponding entry in the array is set to 0 using the update operation \( \text{packet}[n]=0, n++ \), which also increments the value of \( n \) denoting the next packet number to be used. A packet-array entry of 0 for a packet indicates that the packet has been sent into the system. Similarly, as G2 sends packets over channel \( b \), the corresponding entry in the array packet1 is set from 1 to 0 using a similar update operation.

- M1 has been modeled as an automaton with 3 states and 4 transitions as shown in Figure 2.4. When a packet is received over channel \( a \) by the receive operation \( a[p3]? \), the transition is taken from the start location and the control moves to next location. Here, M1 waits to send the packet over channel \( c2 \) to M2. A select statement, \( 'p3:int[0,9]' \) is required to ensure that the automaton M1 selects and synchronizes on the same packet value from 0 to 9 sent by G1. The packet value in the variable \( p3 \) is copied onto the variable \( p1 \) to be used in the subsequent send.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>No time</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Deterministic time</td>
<td>delay 10</td>
<td>delay 1</td>
<td>delay 2</td>
</tr>
<tr>
<td>Non-deterministic time</td>
<td>delay 9</td>
<td>delay 10</td>
<td>delay 11</td>
</tr>
</tbody>
</table>

Table 2.1: Table of delay values

2.2 Uppaal Model without Time

For the tiny two way buffers system with no time, each component of the system has been denoted by a separate automaton in the model. The location marked with red indicates where the control is at the moment. Two global arrays packet and packet1 of the size of the number of packets have been maintained with the initial value for each packet set to 1 indicating that the packet is available.

- G1 and G2 have both been modeled as an automaton with one location and a self loop transition as shown in Figure 2.2 and 2.3. In G1, when the transition is taken, the next value of integer \( n \) (denoting the packet number) is sent over the channel \( a \) by the send operation, \( a[n]! \). As a packet is sent, the corresponding entry in the array is set to 0 using the update operation \( \text{packet}[n]=0, n++ \), which also increments the value of \( n \) denoting the next packet number to be used. A packet-array entry of 0 for a packet indicates that the packet has been sent into the system. Similarly, as G2 sends packets over channel \( b \), the corresponding entry in the array packet1 is set from 1 to 0 using a similar update operation.

- M1 has been modeled as an automaton with 3 states and 4 transitions as shown in Figure 2.4. When a packet is received over channel \( a \) by the receive operation \( a[p3]? \), the transition is taken from the start location and the control moves to next location. Here, M1 waits to send the packet over channel \( c2 \) to M2. A select statement, \( 'p3:int[0,9]' \) is required to ensure that the automaton M1 selects and synchronizes on the same packet value from 0 to 9 sent by G1. The packet value in the variable \( p3 \) is copied onto the variable \( p1 \) to be used in the subsequent send.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>No time</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Deterministic time</td>
<td>delay 10</td>
<td>delay 1</td>
<td>delay 2</td>
</tr>
<tr>
<td>Non-deterministic time</td>
<td>delay 9</td>
<td>delay 10</td>
<td>delay 11</td>
</tr>
</tbody>
</table>

Table 2.1: Table of delay values

2.2 Uppaal Model without Time

For the tiny two way buffers system with no time, each component of the system has been denoted by a separate automaton in the model. The location marked with red indicates where the control is at the moment. Two global arrays packet and packet1 of the size of the number of packets have been maintained with the initial value for each packet set to 1 indicating that the packet is available.

- G1 and G2 have both been modeled as an automaton with one location and a self loop transition as shown in Figure 2.2 and 2.3. In G1, when the transition is taken, the next value of integer \( n \) (denoting the packet number) is sent over the channel \( a \) by the send operation, \( a[n]! \). As a packet is sent, the corresponding entry in the array is set to 0 using the update operation \( \text{packet}[n]=0, n++ \), which also increments the value of \( n \) denoting the next packet number to be used. A packet-array entry of 0 for a packet indicates that the packet has been sent into the system. Similarly, as G2 sends packets over channel \( b \), the corresponding entry in the array packet1 is set from 1 to 0 using a similar update operation.

- M1 has been modeled as an automaton with 3 states and 4 transitions as shown in Figure 2.4. When a packet is received over channel \( a \) by the receive operation \( a[p3]? \), the transition is taken from the start location and the control moves to next location. Here, M1 waits to send the packet over channel \( c2 \) to M2. A select statement, \( 'p3:int[0,9]' \) is required to ensure that the automaton M1 selects and synchronizes on the same packet value from 0 to 9 sent by G1. The packet value in the variable \( p3 \) is copied onto the variable \( p1 \) to be used in the subsequent send.
M1 then sends the value $p1$ over channel $c2$ by the send operation, $c2[p1]!$. On the other hand, automaton M1 on receiving a packet over channel $c1$ from M2 moves to the next location from where it takes the outgoing transition to send the packet over channel $d$ to exit $E1$ using similar operations. The automaton for M2 has similar locations and transitions with the respective channels associated with buffer M2 as shown in Figure 2.5.

- Exits $E1$ and $E2$ have again been modeled as automata with single locations and a self loop transitions as shown in Figure 2.6 and 2.7. $E1$ takes the loop transition on receiving a packet from M1 over channel $d$ by the receive operation $d[p]?$.

  This transition also resets the entry in array $packet1$ for the packet number received to -1 using the update operation `$packet1[p]=-1'`, indicating that the packet has been processed and sent out of the system and should not be reused. Receiving of a packet, as explained above, requires to be accompanied with the select statement.
2.3. Uppaal Model with Deterministic Time

'\texttt{p: int\{0,9\}}' to ensure that the correct packet number from 0 to 9 sent by M1 is chosen.

![Figure 2.6: Automaton for E1 with no time](image)

![Figure 2.7: Automaton for E2 with no time](image)

The system deadlocks when both G1 and G2 send a packet simultaneously which can be seen in Figure 2.8. The deadlock property returns ‘\textit{Property not satisfied} in the Uppaal user interface as shown in Figure 2.9.

![Figure 2.8: Deadlock in model with no time. Locations in red indicate the ones with the system control at the moment of deadlock.](image)

2.3 Uppaal Model with Deterministic Time

Since in the tiny two way buffer system, delays occur in the two generators and the two buffers, the corresponding automata have been assigned local clocks. Also, there is a global clock $z$ that keeps track of the actual time in the system and has been modeled as
an automata with one location and no transitions, and in which the clock keeps ticking continuously. This automaton is shown in Figure 2.10.

Figure 2.10: Automaton for clock

In order to make generator G1 wait for 10 seconds before sending a packet, a local clock variable g1z has been included as shown in Figure 2.11. A guard $g1z=10$ is included on the transition due to which the transition can be taken when the clock value becomes 10. An invariant $g1z < 10$ causes the control to be in the state only when the invariant is true and, hence, the transition has to be taken at time 10. An update of the clock to 0 causes it to restart ticking from 0 after the transition has been taken and the process repeats. Similarly, local clocks have been maintained by G2, M1 and M2 and the required delays have been taken care of by invariants to states and guards to the respective transitions as shown in Figure 2.12, 2.13 and 2.14 respectively. The deadlock in this system occurs after time 10 seconds and can be seen in the automata in the Figure 2.15.
In the tiny two-way buffer system with non-deterministic time, G1 can delay between 9-12 before sending a packet. This is done by including an invariant $g_1z \leq 12$ on the state and a guard $g_1z > 9$ on the transition as shown in Figure 2.16. This ensures that
the control stays in the state not longer than 12 seconds and the transition is enabled at any time at or after 9 seconds. Hence, the transition takes place at a time instant non deterministically chosen in this interval. Deadlock happens even in this system as shown in Figure 2.17.

2.5 Tiny Two-way Buffer System with No Deadlock

In order to study a system with no deadlock, a slightly modified version of the tiny two way buffer system was considered as shown in Figure 2.18. In this system, both
generator G1 and G2 produce a packet after a delay of 10 seconds. However, G1 has an initial shift of 5 seconds after which it begins sending a packet every 10 seconds. Hence, initial delay for G1 is 15 seconds and for G2 it is 10 seconds. Due to this initial shift, both G1 and G2 produce packets alternatively and hence do not end up sending packets at the same time. This system is deadlock free. The automaton for G1 and G2 for this system in Uppaal are shown in Figure 2.19 and 2.20. On testing the deadlock property, the interface displayed the ‘Property is satisfied’ message for this system as shown in Figure 2.21.
Figure 2.21: No-deadlock property satisfied in Uppaal
Chapter 3

Conclusion

The main purpose of this assignment was to perform verification of a certain industrial embedded system. System designs were translated to a formal language that could be used to perform exhaustive verification. Uppaal was chosen due to the ease with which time can be modeled using its concept of timed automata.

Verification of models performed in this project returned implementation bugs and system design issues in the system models in case of the deadlock property and certain other properties, requiring changes to the models and design. Some properties were directly satisfied. The size of the state space for the deterministic models was limited and could be explored. However, when non determinism is applied, the number of states increases drastically. In some cases, the state space became too huge and no verification results could be obtained due to the memory limitations of Uppaal.

3.1 Further Reading

This work includes the complete analysis and verification performed on the industrial system under study. The various goals of the project were carried out sequentially and the experimental results obtained were used for improving the existing models of these machines. However, this work is confidential and hence the entire details of the work cannot be included in this public version of the report. For further details please contact:

- prof.dr. Mark van den Brand
  Chair of Software Engineering and Technology
  Department of Mathematics and Computer Science
  Eindhoven University of Technology
  M.G.J.v.d.Brand@tue.nl
• dr. Suzana Andova
  Assistant Professor
  Department of Mathematics and Computer Science
  Eindhoven University of Technology
  s.andova@tue.nl
References


