Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 407 – 413

ISSN 1896-7094

EMLAN: a framework for model checking of reactive systems software

Artur Krystosik[1]

[1] Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, Warsaw, Poland
a.krystosik@ii.pw.edu.pl

Abstract: Embedded System Modelling Language (EMLAN) is high-level, C-like language for modelling and model checking of embedded, reactive systems. The language addresses a number of topics such as: partitioning of the system, concurrency, interrupts, synchronization mechanisms, time, data transformations, hardware interactions. Model checking of the EMLAN specification is based on translations into DT-CSM (Discrete Time Concurrent State Machines), generation of a reachability graph and checking requirements expressed as CTL temporal formulas. The paper presents the general concepts of verification of reactive systems using EMLAN.

1 Introduction

A range of applications of embedded systems is very wide, from mission-critical systems in aero, medical and automobile industry to popular, home and personal devices. In both areas their reliability is one of the most important factors. Because of concurrency, reactiveness, and interaction with hardware, the correctness of embedded system software is very difficult to prove by testing. One of the most promising ways of embedded system verification is modeling and model checking. Up to date, there are two main research directions. The first one is focused on modeling a system, verification, and then source code generation (manual or automated). The second one is based on the modeling and verification of the existing source code. One of the main problems concerning the first approach is a transformation between the model and the real implementation. The distance between the model and the implementation depends on many factors e.g. level of modeling, understanding the language or methodology by an engineer, applied abstractions and so on. General purpose model checkers, such as SPIN [5], SMV [4] are often used for embedded system verification, however, their usage is not easy, especially for less trained people. Modeling of the various types of concurrency, synchronization mechanisms or interrupts, needs a deep understanding of underlying methodology. In many cases this is a serious problem itself, even without usage of these mechanism in a particular software. To cope with these problems there is proposed a high level, C like language for modeling and verification of embedded system software. The main goal was to create the specification language that is as close to implementation language as possible, with full symbolic model checking possibilities. EMLAN also addresses issues often omitted during software modeling such as: concurrency and time abstractions. The language is based on primitives well known to embedded system engineers such as: task, statement, variable, interrupt, semaphore, monitor and so on. All these features help to reduce the gap between the specification and the further implementation. The paper is organized as follows. Chapter 2 gives the general framework concepts. Chapter 3 contains description of the DT-CSM automata – formalism in which EMLAN is expressed. In the chapter 4 the introduction to EMLAN language is

given. Chapter 5 presents main concepts of model checking of EMLAN specification. Chapter 6 contains conclusions.

2 General concepts

A general concept of verification of embedded systems using EMLAN environment is the following:

- Creating a model of a system software in EMLAN language.
- Translating model into DT-CSM automata.
- Obtaining reachability graph (RG) of DT-CSM automata.
- Reducing the size of the RG by returning from DT-CSM domain, to EMLAN domain (nodes of the reduced graph represents EMLAN instructions).
- Querying the EMLAN-RG using CTL formulas.
- Translating (almost direct) EMLAN specification into program code in C language.

This approach has three main advantages:

- Creating a model is a simple task because of the similarity between EMLAN and C language and because of using high level synchronization and communication primitives.
- Verification is performed in EMLAN domain, using a notions from high-level specification rather DT-CSM specification (however, verification on the automata level is also possible).
- Code generated from EMLAN specification is readable and very close to the model.

3 Discrete Time Concurrent State Machines

Discrete Concurrent State Machines are labeled, directed graphs with integer variables and timers, which can be abstract models of discrete objects, e.g. programs, protocols, hardware devices etc. DT-CSM model is extension of CSM model developed by Mieścicki [7,8]. DT-CSM automaton resembles a well-known Moore finite automaton (or finite state machine, FSM). However, in contrast to conventional FSM, in DT-CSM transitions are labeled with Boolean conditions rather than with symbols from an input alphabet. The transition (s, s') from node s to s', labeled with condition c, means that s' can follow s if (and only if) condition c is true. When s=s' (i.e. an arc makes a 'loop' over the same state) c represents a condition under which the machine can remain in s. Note that two or more conditions can be simultaneously true and – consecutively - more than one arc from a state can be simultaneously enabled. Then, only one of them is selected. The choice is non-deterministic. Note also that arcs labeled with the condition “true” can be used. They are interpreted as spontaneous transitions that require no external events or messages to be enabled. Boolean conditions can be expressed in terms of symbols, variables and timers e.g. a & ! b & (v>5) means that transition is enabled when symbol a occurs, symbol b does not occur at automaton input and value of local variable v is greater than 5.

The key point in the DT-CSM model is that (in contrast to conventional FSM) the sequential occurrence of input symbols is not assumed. Input symbols are not sequenced nor interleaved in any way. They can come either alone or simultaneously or even not come at all. Moreover, any component of a system can transmit its own output symbols that can be inputs to neighboring automata.

The main difference between DT-CSM and CSM lies in introducing integer variables and time. There are two separate sets of variables: local and global. Local variables are accessible only by actions or conditions of automaton to which they belong. All automata can access global variables. Concurrent read of global variables is permitted, but concurrent write is treated as an error. Because no implicit synchronization among automata activities is assumed, the given specification should ensure mutual exclusion during modification of global variables.

Time semantics in DT-CSM is based on notion of discrete time. The passing of time is represented by incrementing of timers, which are defined for a particular automaton. Time is global i.e. each timer is incremented at the same instant of time. Timers can be zeroed by execution of transition actions.

The time semantics of automata system is the following:

- Initial value of all timers is 0.
- Time is passing when all component automata remain in their states (performing transitions to the same state). Time of transitions equals zero.
- When time is passing, each timer is incremented (modulo timer range) when a time predicate is true for a current state of automaton to which a particular timer belongs.

The behaviour of automata system is defined by so-called Reachability Graph (RG), which contains all system states reachable from an initial state. Each system state consist of component automata states, values of all local and global variables and values of all timers. The RG shows all possible configurations or co-incidences of component automata states and all transitions that are likely to occur. The analysis of RG may detect and identify harmful synchronization errors, like a deadlock, a livelock, possible lack of response for some specific event, unwanted simultaneous activity of two components etc. These errors are hardly detectable by simulation and testing, as they may result from very rare coincidences of components' states and external stimuli. In general, RG can be of an enormous size, which causes well-known time and space complexity problems. In practical cases, the number of RG nodes (i.e. system states) can be even of order of 10^20 [2]. To cope with the problem, DT-CSM RG is represented in a form of data structures known as ROBDD [1] that allow for very concise representation.

The inspection of such a large RG cannot be done 'by naked eye'. The typical approach involves the use of temporal logic, where the requirements have the form of temporal formulas [6]. For DT-CSM we use a modified QsCTL temporal logic, which was designed in [3] to investigate RG of CSM automata. On the figure below, there are presented simple automata system and its reachability graph.

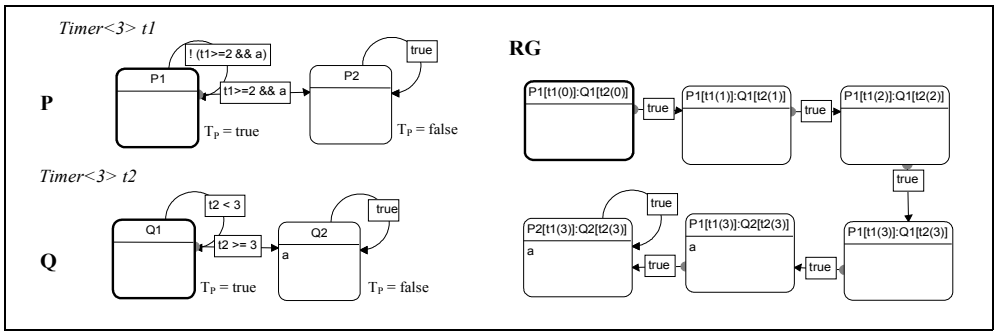

Fig. 1. Sample DT-CSM automata system and its reachability graph

The system consist in two automata P and Q. Each of them has declared a timer. Timers can be incremented in P1, Q1 states only (see time predicates). A transition from P1 to P2 is enabled when timer t1 >= 2 and there is a symbol a at automaton input. Reachability graph of this system shows component automata states, values of timers and symbols generated in states. Note, that states: P1[t1(1)]:Q1[t2(1)] and P1[t1(2)]:Q1[t2(2)] can be replaced with single transition from P1[t1(0)]:Q1[t2(0)] to P1[t1(3)]:Q1[t2(3)] without loosing any information about the system behavior. This technique allows reducing RG size, especially when performed during RG generation.

4 Embedded System Modeling Language

EMLAN is structural language for modeling and verification of embedded systems. It is mainly focused on system software, but interaction with system hardware is also provided. EMLAN assumes some general model of embedded system expressed in a hierarchical manner. On the highest level, the overall model is divided between a model of the system itself and a model of its environment, which is necessary for further verification by a model checking. The environment can be specified either in EMLAN or as set of DT-CSM automata. The process of environment modeling will not be further described because is very similar to system modeling. On the level of the system model we assumed that embedded systems consist of some number of software units and some number of hardware units connected together by some communication means (shared memory, communication channels, interrupts).

The software unit represents elements of the system on which a software tasks can be run e.g. processors, computers. All software units are independent of each other and work in a truly concurrent manner. Tasks inside a software unit can run in truly concurrent, pre-emptive or coroutine mode. The concurrency mode for each unit is selected by a designer according to concurrency mode in a real system or according to applied concurrency abstraction. A concurrency abstraction is widely used technique to reduce a complexity of a model (e.g. pre-emptive, time-sharing systems are often modeled as truly concurrent), however, such decision should be taken after thorough investigation.

The hardware units model this hardware part of the system, which was considered important for verification of system correctness. Because the hardware modeling plays only auxiliary role in our general model, the hardware units can be expressed only as DT-CSM automata.

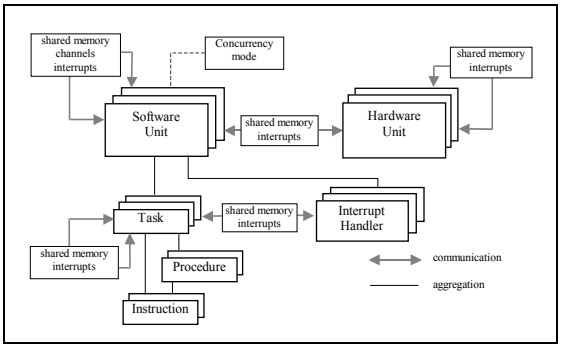

Fig. 2. Components of the EMLAN embedded system model

4.1 Data types

EMLAN has two sets of data types: simple types such as: *int*, *timer*, *enum*, *bool* and so called object types which represent communication and synchronization mechanism e.g. channels, semaphores, mutexes, monitors. Variables of simple types can be declared as: global variables, unit variables and local task variables. Global variables are visible for all tasks in all units, unit variables are visible only for tasks of a given unit and local task variables are accessible only inside a given task. Variables of object types can be declared only on system and unit level, because their usage is limited to communication and synchronization between tasks. The example of variable declaration is given below.

4.2 Tasks and procedures

A task is sequential, structural program, running on a software unit according to selected concurrency mode. Tasks are statically assigned to particular units. Because EMLAN is finite-state model, there is no possibility to dynamically create new tasks. To make the system model more structural EMLAN have the ability to call procedures. The semantic of a procedure is similar to procedures in classical imperative languages. The main differences are, that parameters are always passed by reference and recursive call (direct or indirect) is forbidden. The usage of procedures procedures has no influence on model size, however, increases the readability of the model.

4.3 Statements

Tasks can contain statements of the following types: control statements, arithmetic/assignment statements, interrupt statements, time-passing statements, communication and synchronization statements.

To the control statements belongs *if-else*, *while*, *return*. Boolean expressions for if and while statement can contain variables of simple types, logical operators &, |, !, arithmetic operators +,-, <, >, <=, >= and brackets (,). Semantics of these statements is the same like C language. To the control statement also belongs *switchTo* statement, which suspends the execution of current tasks and resumes execution of another task (passed as the statement parameter). This statement is allowed only for software units, which work in the coroutine mode (process implicitly transfers a control to the other process). A syntax and semantics of arithmetic and assignment statements are also very similar to C language. Assignment statement can contains expressions built from variables of simple types, logical operators &, |, !, arithmetic operators +,-, <, >, <=, >= and brackets (,).

4.4 Synchronization and communication

Synchronization mechanisms are represented by variables, which can belong to the following types: *Semaphore*, *Mutex* and *Condition*. Because the semantics of these mechanisms are well known, a detailed description of these will not be given here.

EMLAN contains one dedicated communication mechanism, which is based on message passing in pure randez-vous mode. This mechanism is represented by type Channel with methods *send* and *recv*. Channels can pass messages, which belongs to simple types. According to randez-vous semantics, both methods are blocking, however, a timeout can be specified or methods can be called inside *select* statement. The select statement provides a selective wait for one or more communication alternatives.

4.5 Interrupts

Interrupts are very often used in embedded systems design and in many cases cannot be omitted during modeling and verification. To cover a wide spectrum of embedded systems, EMLAN interrupt model is defined on two levels: with and without interrupt controller. As it was shown on the figure 3, interrupts form a part of a general model of software unit.

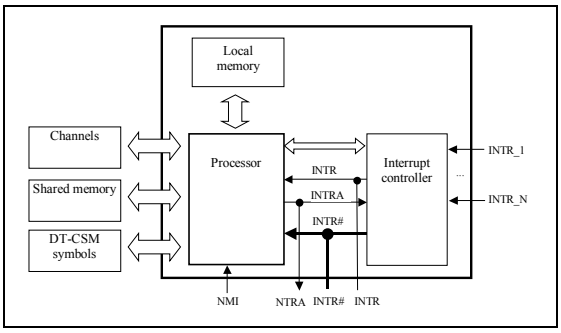

Fig. 3 General model of software unit

Software unit with interrupt controller provides a hierarchical, N-level maskable interrupt mechanism. Interrupts are triggered by edge and stored by IC as long as they are handled. Handling of interrupt on level k blocks all interrupts from levels <1,k>, however, such interrupts are stored by IC and handled later.

Without the interrupt controller, a software units provides a flat interrupt mechanism, with a single lines of rising and acknowledgement of interrupts (INTR, INTRA) and bus on which user puts a interrupt number.

4.6 Time abstractions

EMLAN implements three types of time abstraction: causal, time-independent, discrete time.

In the causal abstraction a time passing is modeled by causal relation between consecutive statements, without possibility to measure any time intervals. DT-CSM automata, in which EMLAN is expressed, imposes to this abstraction a risk of so-called false synchronization. This can result in inconsistency between the model and the real system. The risk can be greatly mitigated by using communication with acknowledgments.

The time-independent abstraction consists in preceding each EMLAN statement with DT-CSM state, which models any possible time passing. This abstraction is very useful, because a system which was verified and recognized correct, is not dependent on the time i.e. the correctness do not depends e.g. on execution speed of its components. The price for this abstraction is increased complexity of a model.

Semantics of discrete time abstraction is the following:

- Time is discrete and measured in time units.
- Time has to be introduced to the given specification directly by timed statements.
- Time passes when all tasks of all software units are blocked on synchronization/communication statements or wait statements.
- All remaining (non blocking) statements do not cause the time to pass.

5 EMLAN model checking

EMLAN is formally expressed by translating the language constructs into appropriate DT-CSM models (automata), which form DT-CSM automata system. The system is translated into ROBDD and next, system reachability graph is obtained. During generation of RG the system invariant checking and detection of concurrent access to global variables are performed. After generation, the RG is investigated using temporal formulas of QsCTL logic. A found flaw is described by counterexample, which is the shortest path from the beginning state to the error state.

The RG graph is expressed in terms of DT-CSM automata, but it can be easily transformed and expressed in terms of EMLAN statements. This allows to express temporal questions at the level of specification language and make easier the analysis of counterexamples, which are then expressed as a path of EMLAN statements.

6 Conclusions

The paper showed main concepts of Embedded System Modeling Language, a language for specification and verification of embedded system software. EMLAN and DT-CSM automata were implemented in EMLAN 1.0 verification tool prepared in ICS WUT. At the end, several conclusions are worthy to be emphasized:

- Specification of ES software in EMLAN is easily understandable and very close to implementation language.
- EMLAN addresses issues often omitted during software modeling such as concurrency and time abstractions.
- EMLAN model is a formal one and can be verified by a symbolic model checking.
- There is a need for further research to fully recognize the possibilities and limitations of EMLAN.

References

1. Bryant R. E. (1995). *Binary Decision Diagrams: Enabling Technologies for Formal Verification.* Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 236-243, 1995.
2. Burch, J. R., Clarke, E. M., and McMillan, K. L. (1991). *Symbolic Model Checking: 1020 States and Beyond.* Proc. International Workshop on Formal Methods in VLSI Design, 1991.
3. Daszczuk W. B. (2003). *Verification of temporal properties in concurrent systems.* Ph.D. thesis, ICS WUT 2003.
4. Halbwachs N., Peled D. (1999). *NuSMV: a new symbolic model verifier.* Proceeding of International Conference on Computer-Aided Verification (CAV'99). In Lecture Notes in Computer Science, number **1633**, pages 495-499, Trento, Italy, July 1999. Springer.
5. Holzmann G. J. (1997). *The Model Checker Spin.* IEEE Transactions On Software Engineering, Vol. **23**, No. 5, May 1997.
6. McMillan K. L. (1993). *Symbolic Model Checking.* Kluwer, 1993.
7. Mieścicki J. (1992a). *On the behavior of a system of concurrent automata.* Institute of Computer Science, WUT, Research Report 20/92.
8. Mieścicki J. (1992b). *Concurrent system of communicating machines.* Institute of Computer Science, WUT, Research Report 35/92.