Using MaxBMC for Pareto-Optimal Circuit Initialization

Sven Reimer  Matthias Sauer  Tobias Schubert  Bernd Becker

Institute for Computer Science
Albert-Ludwigs-Universität Freiburg
79110 Freiburg, Germany
{reimer@sauer|schubert|becker}@informatik.uni-freiburg.de

Abstract—In this paper we present MaxBMC, a novel formalism for solving optimization problems in sequential systems. Our approach combines techniques from symbolic SAT-based Bounded Model Checking (BMC) and incremental MaxSAT, leading to the first MaxBMC solver.

In traditional BMC safety and liveness properties are validated. We extend this formalism: in case the required property is satisfied, an optimization problem is defined to maximize the quality of the reached witnesses. Further, we compare its qualities in different depths of the system, leading to Pareto-optimal solutions.

We state a sound and complete algorithm that not only tackles the optimization problem but moreover verifies whether a global optimum has been identified by using a complete BMC solver as back-end.

As a first reference application we present the problem of circuit initialization. Additionally, we give pointers to other tasks which can be covered by our formalism quite naturally and further demonstrate the efficiency and effectiveness of our approach.

I. INTRODUCTION

In recent years, Bounded Model Checking (BMC) has become a more and more popular technique in the area of formal verification. Unlike traditional Model Checking, in which an entire system gets validated, BMC considers a certain number of temporal steps, starting from a given set of initial states: The implementation is bounded to a given length k, validating whether the property under consideration (i.e., the specification) is satisfied for length k or a counterexample is computed. The length k is incremented until either the system has been unrolled up to a user-defined bound or a witness (i.e., trace) for reaching the negated property under consideration has been found.

Today’s symbolic BMC is predominantly based on SAT solvers as first introduced in [1]. Concerning SAT-based BMC, many accelerating techniques like incremental SAT solving [2] have been developed to speed up the search process and hence the practicability. In that particular case, the basic idea is to reuse already learned information from former SAT solver calls in following similar SAT instances. Likewise, there are several other methods dedicated to the BMC framework [3], [4].

A drawback of the bounded concept in BMC is its incompleteness: Without modifications classical BMC is not able to prove the absence of a witness. Hence, methods have been developed in order to prove the unreachability of the property under consideration. Examples for such approaches are k-induction [5] and Craig interpolation [6].

BMC has a range of applications [7], [8], [9], that in particular includes checking liveness and safety properties. Safety properties tackle the question, whether it is possible to reach a bad state, whereas liveness properties deal with the question whether a good state can always be reached in the system (starting from a set of initial states in both scenarios). However, classical BMC does not allow to specify the quality of traces in the context of a general application. Instead, any trace that requires the least number of unrollings may be returned.

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS) and the Research Training Group “Embedded Microsystems” (GRK 1103).

II. PRELIMINARIES

In this section we briefly introduce the satisfiability problem (SAT), the related optimization problem MaxSAT, and Bounded Model Checking (BMC). The interested reader is referred to [12] for further details.

A. Satisfiability and MaxSAT

Given a propositional formula \( \varphi \), SAT seeks an assignment \( A \) of the Boolean variables \( V \) occurring in \( \varphi \) such that \( \varphi \) evaluates to logic 1. An assignment \( A \) is a function \( A: V \rightarrow \{0, 1\} \). We call \( A \) a model of \( \varphi \) if \( \varphi(A) = 1 \) holds. In that case we also call \( \varphi \) satisfiable. Otherwise, if there is no \( A \) for which \( \varphi(A) = 1 \) holds, the formula is said to be unsatisfiable. Propositional formulae are typically given in conjunctive normal form (CNF), which is a conjunction of clauses. A clause is a disjunction of literals, and
a literal is a variable \( v \in \mathcal{V} \) or its negation \( \neg v \). Therefore \( \varphi \) is satisfied if all clauses are satisfied, and a clause is satisfied iff at least one literal of the clause is assigned to 1.

Today’s state-of-the-art SAT solvers [13], [14] are based on the DPLL algorithm [15], which decides assignments to the variables and deduces resulting assignments (also referred to as implications). A key feature is the concept of conflict driven clause learning (CDCL): in case a clause is unsatisfied due to the assignments (also called conflict), the solver creates a reason in form of a clause for this conflict. We say such a clause is derived from \( \varphi \). The solver resolves the conflict by withdrawing the conflicting assignments and by adding the derived clause to \( \varphi \), which prevents the solver from choosing the same conflicting assignment again.

Many SAT-related formalisms have been introduced in recent decades. One prominent example is the Maximum Satisfiability problem (MaxSAT). Intuitively, in a MaxSAT problem we try to satisfy as many clauses as possible in \( \varphi \). In this context the clauses are also called soft clauses. There are several natural extensions of MaxSAT like Weighted MaxSAT and Partial MaxSAT. In the former extension the clauses are labeled with non-negative weights and the goal is to maximize the sum of the weights of the satisfied clauses. In the latter extension there are additional so-called hard clauses, which must be satisfied, whereas the soft clauses are treated as in MaxSAT. Likewise SAT, one obtains a model which indicates the MaxSAT objective: the number of soft clauses (or the sum of the clause weights) which are satisfied simultaneously.

In the following we always refer to MaxSAT within the meaning of MaxSAT or one of its extensions.

Modern MaxSAT solvers use different techniques for handling the optimization constraints. In general, one can distinguish between three main approaches: branch-and-bound [16], core-guided [17] and iterative [18] algorithms. Common iterative methods for example encode the maximization property of the soft clauses as cardinality constraints and add them to the original formula via adder-, counter-, or sorter-networks [19]. The underlying SAT solver is called iteratively, updating the bounds for the number of satisfied soft clauses with each step. In this paper, we utilize the MaxSAT solver antom [20], applying this iterative approach.

B. Bounded Model Checking

Bounded Model Checking (BMC) is a technique which validates a sequential system by a given exploration limit up to a predefined number of time steps. In particular, BMC considers safety and liveness properties. A common application is to obtain error traces in sequential circuits which are required to falsify a certain safety property.

Intuitively, BMC starts from a fixed initial position (i.e., the set of initial states) and tries to attain a goal within a predefined maximum number of steps. First, it is validated whether the set of initial states already contains the goal. If this is not the case, BMC then checks whether the goal is reachable in one step, in two steps, etc. until either the goal is reached or the exploration limit is exceeded.

The structure of the system and the requested property are encoded as a propositional formula of the form:

\[
BMC_k = I_0 \land T_{0,1} \land \ldots \land T_{k-1,k} \land P_k
\]

(1)

The parts \( I_0 \) encodes the initial states. The terms \( T_{i,i+1} \) represent the so-called transfer function, which is the combinational part of the system. The transfer function defines one sequential step from time frame \( i \) to \( i + 1 \) in the sequential system (e.g. a circuit) under consideration. The predicate \( P_k \) represents the goal, i.e., a property whose reachability after \( k \) steps has to be checked.

If there exists a path in the unfolded system starting at \( I_0 \) and reaching a state satisfying \( P_k \) in \( k \) time frames, \( BMC_k \) is satisfiable. In that case BMC returns a shortest witness satisfying the property. If the property never holds the BMC problem is unsatisfiable.

It can not be proven whether the property is never reachable unless the system is unfolded up to its diameter, and hence the procedure is not complete for all \( k \) less than the diameter (typically, the maximum bound for \( k \) is set far less then the system’s diameter). In recent years some approaches have been presented which are able to show the unreachability of the property and therefore make BMC complete, namely \( k \)-induction [5] and Craig interpolation [6].

Craig interpolation uses the theorem of Craig interpolants [21], which represents an over-approximation of a particular set. Starting with the set of initial states, Craig interpolants are calculated for each transition step in order to obtain an over-approximation of the reachable state set. If the reachable state set does not change in two consecutive time steps a fix-point is reached and if the property is not part of the over-approximation it is proven that it is never reachable, and hence the BMC problem is unsatisfiable.

If the property is part of the over-approximation, either the approximation was too coarse and the procedure is restarted excluding this spurious trace or we have shown that the goal is reachable. For more details of this procedure the interested reader is referred to [6]. In this paper, we make use of the complete BMC solver CIP [22] using Craig interpolants for proving whether the intermediate solution bounds represents a global optimum.

III. MaxBMC

In this section we formalize MaxBMC as an extension of BMC and state an algorithm to solve such kind of problem instances.

A. Definition

The MaxBMC problem is based on the BMC concept, i.e., it also asks whether a safety or liveness property in a sequential system holds within a predefined number of time steps. Additionally, if a witness is found, MaxBMC asks for the quality \( q \) of this witness. In particular, the witness with the highest quality for each transition step is identified, and therefore, MaxBMC determines the Pareto-optimal qualities for the sequence lengths until a bound \( k \).

As in BMC, one can encode the structure of MaxBMC as CNF, including soft clauses:

\[
\text{MaxBMC}_k = I_0 \land T_{0,1} \land \ldots \land T_{k-1,k} \land P_k \land O_k
\]

(2)

The parts \( I_0, T_{i,i+1} \) and \( P_k \) are defined as in Eq. 1. In extension to classical BMC we consider two properties for each unrolling depth. The first one is a property \( P_k \) describing requirements that need to hold in order to form a valid solution. The second objective \( O_k \) is a symbolic representation of any optimization problem which can be translated to MaxSAT. \( O_k \) consists of clauses that describe the quality of a witness (i.e., the soft clauses of the encoded MaxSAT problem) and provides the soft clause interface. Hence, the number of satisfied soft clauses are directly mapped to the quality.

Intuitively, we ask whether a property is reachable within a given bound \( k \) and if it is reachable we determine its quality. The quality is given by the MaxSAT objective of \( \text{MaxBMC}_k \), i.e., the sum of the satisfied soft clauses in \( O_k \). We denote the result of the optimization problem and therefore the quality of the best witness with sequence length \( k \) as \( q_k \). The quality of witnesses with different lengths indicate solution bounds. Therefore we define a lower \( (q_{\text{low}}) \) and upper \( (q_{\text{high}}) \) bound of the optimization results \( q_i \) with \( 0 \leq i \leq k \).

In MaxBMC one can prove the reachability of the property in the same manner as described in Sec. II-B. Additionally, in MaxBMC we have to consider the case that for some depth \( i \) the property is reachable, but for a depth larger \( i \) it is not. This case is omitted in BMC, since one is only interested in the shortest trace to the good/bad state. This proof can be done quite similar to the standard reachability check by adding an additional constraint forcing a trace with at least depth \( i + 1 \).

Furthermore, we extend the concept of proving reachability of a property to the question whether the solution bounds of the
secondary objectives can be extended. Therefore one has to apply a solution bound proof, demanding an optimization result $q^*$ which is not within the solution bounds of $q_{low}$ and $q_{high}$. If such a solution does not exist, we have shown the unreachability of a property, demanding an optimization quality $q^*$ and hence obtained a proof that the solution bounds can not be improved. In the following sections we state more details of the composition of such a proof.

### B. Algorithm

In this section we state the MaxBMC algorithm, which 1) computes the optimization results for each iteration and 2) proves the optimality of the solution bounds.

Fig. 1 shows the algorithmic flow, where the blue parts indicate steps involving a complete BMC solver, and the orange parts show the integration of a MaxSAT solver. An illustrating example is given in Fig. 2.

Validating $MaxBMC_k$ requires that the property $P_i$ holds, which is always checked, before we can tackle the optimization problem. Starting with a depth of $i = 0$, it is checked whether $P_i$ holds (in the initial states). In the example, the property $P_0$ is already satisfied, but in the general case it may not hold. In such situations we validate whether the property is (still) reachable beyond the current depth using a complete BMC solver. If the property does not hold the currently identified solution bounds are returned as no better solution can exist. Otherwise, a trace of minimal length to the next depth larger $i$ where the property holds is generated. Hence, we update the depth $i$ and continue with the corresponding time step.

At this point, a depth $i$ is identified, where the property $P_i$ holds and we determine the optimization objective for this depth using an iterative MaxSAT approach (c.f. II-A). We commit the BMC part of Eq. 2 as hard clauses and add the symbolic representation of $O_i$ to a MaxSAT solver. The solver returns a witness at depth 4 and the flow continues with checking $P_4$. When the proof is executed again at depth 16 (some steps are skipped in the example), the optimal bounds can be shown and the final solution bounds $q_{low} = 2$ and $q_{high} = 12$ are returned.

The algorithm is sound in the sense that 1) for every unrolling depth $i \leq k$ the optimization result $q_i$ is calculated in case the property $P_i$ holds and 2) the maximum solution bounds are given by $q_{low}$ and $q_{high}$ if the optimality of the bounds could be proven.

For the second part we need to validate that our solution bound proof is sound. Since we use an iterative MaxSAT approach for the optimization constraint, $O_i$ is encoded as clauses which are added to the combined property. An accordant iterative MaxSAT solver would encode an additional network for the cardinality constraints for the soft clauses and iteratively add constraints bounding the optimization result via the network. Here, we add the network encoding as part of the property and bound the property (i.e., the number of satisfied soft clauses) by triggering $< q_{low}$ or $> q_{high}$ implicitly as part of the combined property $OP_k$. Therefore, $OP_k$ defines a safety property asking whether we reach a certain property ($P_k$) at which the optimization result is below $q_{low}$ or above $q_{high}$. The BMC solver will return such a witness with the shortest sequence length if it exists. Otherwise the BMC problem is classified as unsatisfiable.

The algorithm is complete in the sense that 1) it detects whether $P_k$ is reachable and 2) returns the optimal solution bounds. Again, the first part is clear, since a complete BMC solver checks whether $P_k$ is reachable and therefore our algorithm is complete, too. This also holds in the case the property is reachable only until a time step $i$ by forcing a solution with more than $i$ time steps.

*Figure 2. Illustrating MaxBMC example*
If the user-given bound \( k \) is large enough also the second part of the completeness holds. Since the solution bound proof is sound and the underlying procedure to check these bounds is complete, also the optimality of the solution bounds can be guaranteed.

C. Extensions

Based on the basic MaxBMC formalism defined in Eq. 2 one can define several natural extensions, depending on the specific requirements of an application.

One may extend the number of optimization objectives checked per time frame, i.e., there are possibly multiple \( O_k \)'s in Eq. 2 that can be optimized together or separately. In the former case we obtain one combined quality \( q \) we obtain one combined quality

Another straightforward extension is the definition of weights associated with each soft clause leading to Weighted MaxBMC likewise Weighted MaxSAT. In this case, the quality of a Weighted MaxBMC instance is given by the sum of the weights of the satisfied soft clauses in \( O_k \).

IV. Implementation details

In this section we describe the underlying solver technologies we used and the modifications we made in order to solve the MaxBMC more efficiently.

A. Incremental MaxSAT

As shown in Section III, solving MaxBMC instances necessitates solving a series of similar MaxSAT problems. After each iteration, additional transition relations and further optimization constraints are added. Parts of earlier iterations are removed, but some stay unchanged and may still contain useful information. Likewise classical BMC with incremental SAT solving, MaxBMC may also benefit from an incremental MaxSAT solver which allows re-usage of information gained while solving previous MaxSAT instances. Incremental MaxSAT allows to change optimization constraints which necessitates adaptations to the solving process that go beyond classical incremental SAT.

The core of our incremental MaxSAT implementation is our in-house SAT solver antom [20], which supports incremental SAT solving and provides a MaxSAT interface. Internally, the maximization problem is expressed by a sorting network [23].

We modified the solver to include the following techniques known from the BMC context: Constraint Sharing [3] and Constraint Replication [4]. The reader is referred to the related references for more details of the methods.

Constraint Sharing and Constraint Replication are techniques for reusing learned clauses from former SAT solver calls. Using incremental solving in BMC, only one SAT instance, which changes with every call of the solver, is used for all calculations. In particular, parts of the formula are removed (e.g. by adding the transition relation \( T_{i,i+1} \) and related property \( P_{i,i+1} \), the property \( P_i \) has to be removed), and thus derived clauses originating from removed parts are invalid in later solver calls. In general, derived clauses from former solver calls are more beneficial for the BMC approach than using a single SAT instance for each transition step separately. In BMC the issues with invalid clauses in incremental solving are handled by adding trigger literals to the clauses which define the property.

The trigger literals allow to (de-)activate parts of the formula, which have to be removed in a later solver call. We call the clauses of these parts the temporary clauses of \( P_i \). Assume a trigger variable \( t_i \) for time frame \( i \). Then each temporary clause \( c = \langle t_{i_1} \lor \cdots \lor t_{i_n} \rangle \) from time frame \( i \) is extended to \( c = \langle t_{i_1} \lor \cdots \lor t_{i_n} \lor \neg t_i \rangle \). Thanks to the concept of assumption-based solving and the learning mechanism in modern SAT solvers all clauses which are derived by at least one temporary clause will also contain the trigger literal and hence are also marked as temporary clause. As an example in BMC consider a SAT solver call for transition step \( i \). Adding the assumption \( \neg t_i \) activates the temporary clauses (i.e. clauses describing the property \( P_i \)) of step \( i \). At the same time the temporary clauses of the former transition step \( i-1 \) are deactivated by adding a clause containing only the trigger literal \( t_{i-1} \). This will satisfy all temporary clauses with this trigger literal, and hence the property \( P_{i-1} \) is not constraining anymore.

The concept of trigger literals can be adapted likewise to incremental MaxSAT: the clauses of the properties \( P_k \) are (de-)activated as in BMC. Furthermore we have to consider clauses which are used for the encoding of the optimization constraints. We have to deactivate this encoding after each transition step, since the MaxSAT instance is only valid for the current one. Thus, the optimization constraints of transition step \( i \) have to be treated as temporary clauses for \( i \). The trigger literal is added for these clauses in order to (de-)activate them.

Moreover, we have to take care of the constraints defining the optimization bounds added during a single MaxSAT solver call. If a classical iterative MaxSAT solver finds a bound for the optimization result, a new hard clause is added to the underlying SAT solver constraining this bound. Adding this bounding constraint is not sound anymore in an incremental approach since the bounds and any derived clauses originating from this constraint are only sound for the current transition step. Hence, instead of bounding the result by adding a clause, in incremental MaxSAT we add the bounds as assumption to the problem. This ensures that the constraint only holds for this time step and that any derived clause resulting from this constraint will contain the assumption literal, i.e., it is marked as a temporary clause and therefore we are able to deactivate these clauses.

In case \( q_k \) increases monotonically, i.e., the optimization result of a transition step is either equal or better than the result of the former ones, one can reuse these bounds. In particular, if we have determined an optimization result \( q_i \) in transition step \( i \) we can commit \( q_i \) as a minimum bound for all following transition steps to the MaxSAT solver. This can be done accordingly with monotonic decreasing \( q_k \). In our experiments (c.f. Sec. VI) we observed that this extension is very beneficial for the MaxSAT solver. Typically, the optimization goal describes a mandatory part of the whole MaxSAT instance as the encoding of the cardinality constraints are quite expensive. This encoding can be largely simplified by adding the additional bounding constraints.

To the best of our knowledge, these extensions lead to the first incremental MaxSAT solver. The authors of [24] propose that the usage of incremental MaxSAT would be beneficial for their purpose, but it is presumably not implemented. By applying this solver, our algorithm is able utilize an incremental core algorithm as in BMC with incremental SAT solving and therefore profits from learned information to speed up the solving process.

B. Solution bound proof

We use the in-house BMC solver CIP [22], which supports unreachability proofs by Craig interpolation, as a back-end solver in order to derive on the one hand the proof whether the property \( P_k \) is reachable and on the other hand whether the solution bounds for the optimization problem can be improved. As described in Sec. III-B we have to encode the cardinality constraints of the MaxSAT problem. An additional constraint is added, demanding \( q^* \) original soft clauses to be satisfied, where \( q^* \) is either below \( q_{low} \) or above \( q_{high} \). For example, consider the situation that we are currently applying time step \( i \) and \( q_{high} \) was not improved for a user defined threshold of time steps. Now, we call CIP, encoding the safety property: \( P_i \) holds and the quality of the optimization objective is higher than \( q_{high} \). If there exists such a solution, CIP will return the witness with its sequential depth where the safety property is violated. Then we can update our result for \( q_{high} \) and proceed with our main MaxBMC loop. Otherwise, the BMC solver was able to prove that the quality is not above \( q_{high} \) for any sequential depth. Hence, we are able to fix \( q_{high} \) as the maximum optimization solution. This procedure is done analogously for \( q_{low} \). If both \( q_{low} \) and \( q_{high} \) are fixed our algorithm terminates.
V. APPLICATIONS

In this section we present applications that can be formalized as MaxBMC problems. First, we briefly introduce the problem of circuit initialization which also serves as a reference problem in the experimental results section. Second, we give some pointers to further applications which can be translated into our formalism.

A. Circuit initialization

The problem of initializing (or synchronizing) a given circuit is a well considered problem in the area of testing [10], [11] and is closely related to state reachability problems known from classical BMC. The problem handles the question whether a sequential circuit is initializable, i.e., all flip-flops can be set to a known value, starting from the completely unknown state. Since many circuits are not completely initializable due to their internal circuit function, there is a high interest in finding the maximal subset of flip-flops that can be initialized.

This problem can be formalized as a MaxBMC problem, where the number of initialized flip-flops can be seen as the quality of a trace, i.e., the optimization goal. To represent unknown values, the transfer function $T_{i,i+1}$ is encoded using 01X logic [25]. The maximization goal $q_k$ is given by the number of flip-flops which can be initialized simultaneously. Hence, a maximization over these literals leads to the sequence that initializes as many flip-flops as possible. Note that the property $P_k$ is always trivially true, since each solution to the underlying BMC instance represents a valid sequence even if no flip-flop gets initialized.

Another application area is the optimization of the dynamic power consumption of a sequential circuit [27] which can be estimated by the switching activity of circuit lines, i.e., the amount of times a certain line switches its logic value from 0 to 1 and vice versa. This switching activity can be defined as quality for MaxBMC. The resulting instance is able to identify traces in the system with minimum/mass maximum power consumption in order to optimize the design of the system.

Apart from the circuit domain there are extensions of the well-known planning problem: the question of cost-optimal planning or preference-based planning. In contrast to classical planning one seeks for secondary criteria for a plan (e.g., minimal sequence length, minimal action costs, or maximal state rewards). In [28], [29] methods are proposed to find such plans using Weighted MaxSAT in order to tackle the optimization criteria. The authors consider the construction of the plan detached from solving the MaxSAT problem and no incremental usage of a MaxSAT solver is applied. The plan construction can be formalized as satisfiability problem [30] and we suppose that these problems can be naturally translated into MaxBMC. A plan computation can be seen as traversing transition relations $T_{i,i+1}$, where the planning goal is a property $P_k$ and the optimization criteria can be represented by different qualities $q_k$, translated into $O_k$.

In general our formalism covers problems containing a transition system together with cost functions, where the goal is to find a trace within the system with minimum/maximum costs. Additionally, we allow constricting properties, which have to hold independently from the cost-optimization goal.

VI. EXPERIMENTAL RESULTS

We considered the circuit initialization problem as described in Sec. V-A. Therefore we used sequential versions of commonly used academical and industrial benchmark circuits. All measurements were performed on a single core of a 3.3GHz Intel Xeon processor with a time out of 4 CPU hours per circuit.
We state a sound and complete algorithm which is also able to yield the best possible solution bounds for each possible time step. Additionally, we developed an incremental MaxSAT approach by leveraging techniques from incremental SAT for BMC. Furthermore, we want to investigate in applying other techniques for solving Weighted MaxBMC where ILP formulations bound and methods based on ILP solvers. This is of particular interest for solving MaxBMC approach outperforms previous methods (e.g., [32], [33]) in BMC and providing the CIP solver and Jürgen Schlöffel (Mentor Graphics Hamburg, formerly NXP) for supplying industrial benchmarks.

VII. Conclusion

We presented MaxBMC, a formalism for defining optimization problems in sequential systems, optimizing secondary objectives. We state a sound and complete algorithm which is also able to yield the best possible solution bounds for each possible time step. Additionally, we developed an incremental MaxSAT approach by leveraging techniques from incremental SAT for BMC.

We express the problem of finding initialization sequences as MaxBMC and give some pointers to further applications. Experimental results demonstrate the effectiveness and applicability of our MaxBMC solver.

As future work we plan to investigate the usage of alternative incremental approaches for solving MaxSAT such as branch-and-bound and methods based on ILP solvers. This is of particular interest for solving Weighted MaxBMC where ILP formulations tend to be more beneficial than iterative MaxSAT approaches. Furthermore, we want to investigate in applying other techniques for providing the solution bound proofs.

Acknowledgements

The authors like to thank Stefan KupferSchmid for his insight in BMC and providing the CIP solver and Jürgen Schöflöffel (Mentor Graphics Hamburg, formerly NXP) for supplying industrial benchmarks.

References