Design Optimization for Hardware-Based Message Filters in Broadcast Buses

Lea Schönberger, Georg von der Brüggen, Horst Schirmeier, and Jian-Jia Chen
TU Dortmund University, Dortmund, Germany
{lea.schoenberger, georg.von-der-brueggen, horst.schirmeier, jian-jia.chen}@tu-dortmund.de

***Abstract*—In the field of automotive engineering, broadcast buses, e.g., Controller Area Network (CAN), are frequently used to connect multiple electronic control units (ECUs). Each message transmitted on such buses can be received by each single participant, but not all messages are relevant for every ECU. For this purpose, all incoming messages must be filtered in terms of relevance by either hardware or software techniques. We address the issue of designing hardware filter configurations for clients connected to a broadcast bus in order to reduce the cost, i.e., the computation overhead, provoked by undesired but accepted messages. More precisely, we propose an SMT formulation that can be applied to i) retrieve a (minimal) perfect filter configuration, i.e., no undesired messages are received, ii) optimize the filter quality under given hardware restrictions, or iii) minimize the hardware cost for a given type of filter component and a maximum cost threshold.**

***Index Terms*—automotive, controller area network, broadcast bus, message filtering, design optimization**

I. Introduction

In the field of automotive engineering, broadcast buses, e.g., CAN [1] and LIN [2], are frequently used to connect multiple electronic control units (ECUs) and transmit data. *Messages* are not sent to a specific receiver node but written to the bus and thus received by every node connected to the network. However, not all messages are relevant to all receivers, wherefore each node must determine if an incoming message is *desired* or *undesired* by a *filtering solution*. Regarding automotive applications, a merely software-based filtering solution is preferably avoided, especially for ECUs running at low clock speeds, since each message received by a network node triggers an interrupt and must be further processed [3]. This computation overhead can be reduced or completely avoided applying hardware filters, since undesired messages are passed to the software infrequently or not at all, depending on the actual *filter configuration*.

A *correct* hardware filtering solution accepts all desired messages, while a *perfect* hardware filtering solution accepts all desired messages and blocks all undesired messages. When an undesired message is not filtered out by hardware, a software filtering solution must be involved. Here, the resource consumption must be taken into consideration to guarantee the system's operability. An *imperfect* hardware filtering solution accepts all desired messages and some undesired messages to be further handled by a software filtering solution. *Imperfect* hardware filtering solutions are not preferred though not always avoidable due to the limited number of hardware filters in commercially available hardware components.

The challenge of designing hardware message filters for broadcast buses is addressed quite scarcely. Most work regarding broadcast buses rather focuses on the priority assignment as well as on schedulability analysis, especially with respect to the CAN bus, e.g., [4]. However, the issue of hardware message filter design is restricted neither to CAN nor to the automotive sector, but can be encountered in any scope of application where broadcast buses are used. Notwithstanding, to the best of our knowledge, no general approach to this problem has been proposed so far. The first work particularly examining the optimization of hardware filtering has been published by Pölzlbauer et al. [3] in 2017 and specifically addresses CAN. They introduce a quality metric for hardware filtering solutions that quantifies the penalty due to undesired but accepted messages, denoted *Quality of Filter (QoF)*, which is given as $QoF = \sum_{m \in M^U} \frac{[\text{m is accepted}]}{T_m}$ where M^U is the set of undesired messages, T_m is the period of a (periodic) message $m \in M^U$, and $[P]$ is the Iverson bracket. The *QoF* is optimal if the expression given above evaluates to zero, i.e., the filter is perfect. There are two distinct problems regarding the configuration or design of hardware filtering solutions: i) At design time, i.e., under no hardware limitations, the number of undesired messages accepted by the filter(s) may be reduced to zero by employing a suitable number of hardware filters. ii) For an existing platform, minimizing the *QoF* is of utmost interest to reduce unnecessary overhead for handling undesired (but accepted) messages via software. Here, we deal with both problems and make the following contributions:

- We propose a *Satisfiability Modulo Theories (SMT)* [5] formulation to retrieve a perfect filter configuration, if existent, under no hardware limitations.
- We prove that the obtained perfect filter is minimal for a specific filter component type.
- For a given hardware, we provide an SMT formulation to find an imperfect filter configuration.
- Moreover, we demonstrate how to minimize the hardware cost of a filter configuration for a given type of filter component and a maximum cost threshold.
- We evaluate our approaches in an industrial case study.

II. System Model and Filtering Process

Consider a system comprising a set of nodes, e.g., ECUs, which are connected by a single broadcast bus such as CAN [1]. Each node pertaining to the network broadcasts messages indicated by a number of predefined IDs, which

978-3-9819263-2-3/DATE19/©2019 EDAA

Table I: A filter (one mask, one tag per mask) and a number of exemplary messages.

	bit pattern	accepted?	bit pattern	accepted?
mask	011 1111 1110			
tag	000 1100 1001			
filter	∗00 1100 100∗			
ID	000 1100 0001	no	100 1000 1000	no
ID	000 1100 1000	yes	100 1100 1000	yes
ID	000 1100 1001	yes	100 1100 1001	yes

may be received by all other network members. We denote this set of prespecified message IDs (also referred to as the set of prespecified messages) M. A broadcast bus message m is characterized by a tuple (b_m, T_m), where b_m indicates the unique message ID – a bit vector of length H, which is typically used to specify the message priority, – and T_m denotes the minimum inter-arrival time or period. We omit all other parameters, since these are not relevant here.

Each node connected to the bus has a set of buffers which can either send or receive messages. Since only a subset of all messages is relevant to each node, the overall set of messages M is partitioned into *desired messages* M^D and *undesired messages* M^U with $M = M^D \cup M^U$ and $M^D \cap M^U = \emptyset$ for each node. To specify M^D, a node has a set of *filters* $F = f_1, \ldots, f_n$, also denoted *filter configuration*. Each filter f is characterized by a tuple x_i, Y_i, where x_i is a *mask* with $i \in \mathbb{N}$ and Y_i is a set of *tags* $y_{i,k} \in Y_i$ with $k \in \mathbb{N}$. Both masks and tags are bit vectors of length H that are matched with a message ID b_m. Since b_m is unique, the patterns represented by a mask x_i and its tags $y_{i,k}$ can be modified so that they correspond only to a selected set of message IDs. Hence, a message m is accepted by a filter f consisting of a mask x_i with tag $y_{i,k}$, if and only if the statement $b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)$ is true, where b_and is the bit-wise and operation. An example is given in Table I portraying a filter consisting of one mask with one tag as well as a number of message IDs. The filter pattern resulting from the given mask and tag can be understood as follows: Whenever a bit is marked as irrelevant, i.e., set to 0 in the mask, it is represented by a *don't care*, denoted $*$. All remaining bits indicate the values required from a message ID to be accepted.

III. Perfect Filters

In order to obtain a *perfect* filter configuration, if existent, we establish an SMT formulation. SMT solvers can be applied to determine if a certain propositional formula is satisfiable not only under general first-order logic but also with respect to a certain background theory specifying the exact interpretation of particular symbols. Due to the fact that each message as well as each filter can be expressed by Boolean algebra, a number of assertions can be formulated specifying the actual requirements a filter must satisfy as well as a given maximum *QoF*. If a model, i.e., a set of variables, exists for which the given assertions are satisfied, one possible solution can be obtained by means of an SMT solver. However, the derived solution is not necessarily the only one.

A. Accepting Desired and Nonexistent Messages

Since for a perfect filter accepting or rejecting unspecified messages has no impact on its *QoF*, we permit each filter to accept any $m \in M^{ok}$, where M^{ok} is defined as $M^{ok} = M^{all} \backslash M^U$ and M^{all} is the complete set of possible message IDs. This may lead to a reduction in the number of filters required to achieve perfect filtering. For a given number of masks x_i with tags $y_{i,k}$ we propose the following formulation:

$$\forall m, i, k, \quad z_{m,i,k} = (b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)) \quad (1)$$

$$\forall m \in M^D, \quad \vee_i \vee_k \; z_{m,i,k} = true \quad (2)$$

$$\forall m \in M^U, \quad \vee_i \vee_k \; z_{m,i,k} = false \quad (3)$$

Eq. (1) describes the filtering process according to which a particular message m is either accepted or blocked by a filter with mask x_i and tag $y_{i,k}$, while Eq. (2) and Eq. (3) ensure the perfectness. Evidently, each filter configuration retrieved under these specifications, if existent, is a perfect filter configuration.

B. Filtering with Prime Implicants

To obtain a *minimal* perfect filter configuration, prime implicants can be used, which can be generated by merging two or more minterms, e.g., by the well-known algorithm of Quine & McCluskey [6]. We extend our SMT formulation for the case that each mask x_i has only one tag $y_{i,k}$, i.e., $k = 1$, by Eq. (7), stating that each filter $f \in F$ must be a prime implicant of M^{ok}, i.e., $f \in PI(M^{ok})$.

$$\forall m, i, k, \quad z_{m,i,k} = (b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)) \quad (4)$$

$$\forall m \in M^D, \quad \vee_i \vee_k \; z_{m,i,k} = true \quad (5)$$

$$\forall m \in M^U, \quad \vee_i \vee_k \; z_{m,i,k} = false \quad (6)$$

$$\forall i, k, \quad b\_xor(x_i, y_{i,k}) \in PI(M^{ok}) \quad (7)$$

Theorem 1 (Minimal Perfect Filter Configuration)**.** *Any filter configuration F obtained applying Eq.* (4)–(7)*, where each mask x_i has one tag $y_{i,k}$, i.e., $k = 1$, is perfect and minimal.*

Proof. F being perfect results directly from Eq. (6).

To show that F is minimal, consider a non-minimal, perfect filter configuration $F' = f_1, \ldots, f_n$. Since F' is not minimal, at least two filters $f'_a, f'_b \in F'$ can be reduced to a filter f_c by substituting at least one filter bit by a *don't care*, so that f'_c accepts all messages accepted by f_a and f_b but blocks all undesired messages $m \in M^U$. Assume that we proceed to reduce F' as far as possible while maintaining its optimal *QoF*, i.e., we transform F' from a non-minimal perfect filter configuration to a minimal perfect filter configuration. Since F' cannot be further reduced without accepting any $m \in M^U$, it is clear that $F' \subseteq PI(M^{ok})$ with $M^{ok} = M^{all} \backslash M^U$. □

Please note that the generation of prime implicants may become computationally intractable depending on the size of the message space, which depends on the predefined message ID length H. As an alternative, the SMT formulation proposed in Sec. III-A can be applied if no limitations regarding the available hardware components exist. Otherwise, an *imperfect* filter can be used.

IV. Imperfect Filters

Since hardware components come with certain restrictions, a perfect filter configuration cannot always be achieved, so that it is inevitable to accept a number of undesired messages. However, not all undesired messages induce the same cost: Consider two messages $m_a, m_b \in M^U$, where m_a is transmitted frequently and m_b rarely. Evidently, m_a must be revised by a software filtering solution more often than m_b and thus is more costly. Accordingly, the set of undesired but accepted messages should be chosen such that the QoF is minimized.

A. Minimizing the QoF

Aiming to minimize the *QoF* of a given hardware solution, i.e., a given number of masks x_i with a given number of tags $y_{i,k}$, we extend our SMT formulation:

$$\forall m, i, k, \quad z_{m,i,k} = (b\_and(x_i, y_{i,k}) = b\_and(x_i, b_m)) \tag{8}$$

$$\forall m \in M^D, \quad \vee_i \vee_k \; z_{m,i,k} = true \tag{9}$$

$$\forall m \in M^U, \quad \left(c_m = \frac{1}{T_m}\right) \oplus (\vee_i \vee_k \;\; z_{m,i,k} = false) \tag{10}$$

$$\forall m \in M^U, \quad (c_m = 0) \oplus (\vee_i \vee_k \;\; z_{m,i,k} = true) \tag{11}$$

$$\sum_{m \in \mathbf{M^U}} c_m \leq c_g \tag{12}$$

Each $m \in M^U$ must be either rejected by all filters or to induce a cost greater than zero by Eq. (10). Correspondingly, Eq. (11) enforces each $m \in M^U$ to induce either zero cost or to be accepted by at least one filter. In Eq. (12), a cost threshold c_g is set that must not be exceeded. More precisely, c_g is initialized with an upper bound that is stepwise reduced during the optimization process applying, e.g., simple heuristics such as binary search, random search etc.

B. Minimizing the Hardware Cost

Although status quo hardware usually cannot be modified or exchanged, we do not omit the issue of minimizing the hardware cost during the message filter configuration design process. When a new platform is developed, it may be sensible to reduce the number of required filter components to lower the overall system cost as well as the system size. Especially for low-budget embedded devices, both factors may be non-negligible and thus allow for a certain amount of computational overhead. Here, the SMT formulation in Sec. IV-A can be applied, where c_g is a fixed upper-bound on the tolerable *QoF*. In contrast to the previous scenario, only the number k of tags per mask but not the number i of masks is fixed here, since the considered hardware component type must be given. To determine the minimum required number of hardware components, i is decreased in each iteration until the cost threshold c_g is overshot.

V. Evaluation and Industrial Case Study

In the following, we validate our previously suggested approaches with a case study based on industrial benchmarks. For this purpose, we first outline the considered benchmarks as well as the actual experiment setup, before we discuss the results.

Table II: The modified SAE benchmark by Lesi et al. [8] with randomly chosen desired messages and synthetic message IDs generated according to a uniform distribution.

ID [hex]	T [ms]	Des.	ID [hex]	T [ms]	Des.	ID [hex]	T [ms]	Des.
0x058	20	yes	0x28A	20		0x5B4	5	yes
0x06B	20		0x2AD	20	yes	0x5CE	10	
0x085	5	yes	0x2C6	1000		0x601	1000	
0x0C6	20	yes	0x2EF	10		0x624	20	
0x0DC	20		0x309	20	yes	0x644	10	
0x11E	5		0x316	10		0x667	20	
0x138	1000		0x336	20		0x66B	20	yes
0x13F	20	yes	0x351	20	yes	0x6AB	20	yes
0x148	20	yes	0x364	20	yes	0x6D2	20	
0x15E	20		0x38A	20		0x6D3	20	yes
0x1BA	5		0x3A1	20		0x6F4	20	yes
0x1F8	20		0x3F7	20		0x6F8	100	
0x209	20	yes	0x40A	5		0x725	10	
0x212	1000		0x479	1000		0x755	20	
0x235	10		0x4B4	5		0x772	20	
0x236	20		0x4B5	20	yes	0x77C	5	yes
0x23F	20	yes	0x564	20		0x7D8	1000	

A. Experiment Setup and Implementation Details

In our case study, we consider two different industrial benchmarks. The first benchmark is the network specification of the battery electric vehicle *epsilon*'s (http://epsilon-project.eu/) heating, ventilation, and air conditioning controllers (denoted *epsilon* benchmark) provided by Pölzlbauer et al. [3]. It comprises 55 11-bit message IDs with periods, out of which 11 are desired and 44 undesired. Second, we examine the SAE benchmark for the CAN bus [7] in a modified version by Lesi et al. [8] comprising 51 messages. From this set, we select a number of 18 desired messages. Since message IDs are not provided, we randomly create 11-bit message IDs in a range from 0 to 2047. The resulting set of message IDs is given in Table II. In addition, we generate a number of synthetic sets of message IDs and randomly assign periods from the set $\{50ms, 100ms, 500ms, 1000ms\}$.

We conduct our experiments on a machine with Intel® Xeon® X5650 processor comprising 6 cores (2.67GHz, 2 virtual threads per core) and 20 GiB RAM. To evaluate our SMT formulation, we employ the Z3 solver by Microsoft [9]. In our experiments, we focus on different aspects. First, we determine a minimum perfect filter configuration for the epsilon benchmark for a filter component with one tag per mask. Thereon, we reduce the number of filters stepwise to 1 and report the resulting *QoF* and the time required to find a solution. We compare our findings to the results reported in the work of Pölzlbauer et al. [3]. For comparison, we examine the SAE benchmark under the same settings. Second, we analyze the impact of the number of tags per mask on the time required to obtain a solution. Due to space limitation, we only consider the epsilon benchmark here. Furthermore, we consider the hardware cost optimization, i.e., minimizing the number of required hardware components for a given *QoF* threshold (cf. Sec. IV-B), wherefore we consider a component with 1 tag per mask and a set of 100 message IDs, of which 30 are desired for a permitted overhead of 5%, 10%, and 20% of the maximum possible *QoF*. For each setting, we generate 5 synthetic message sets.

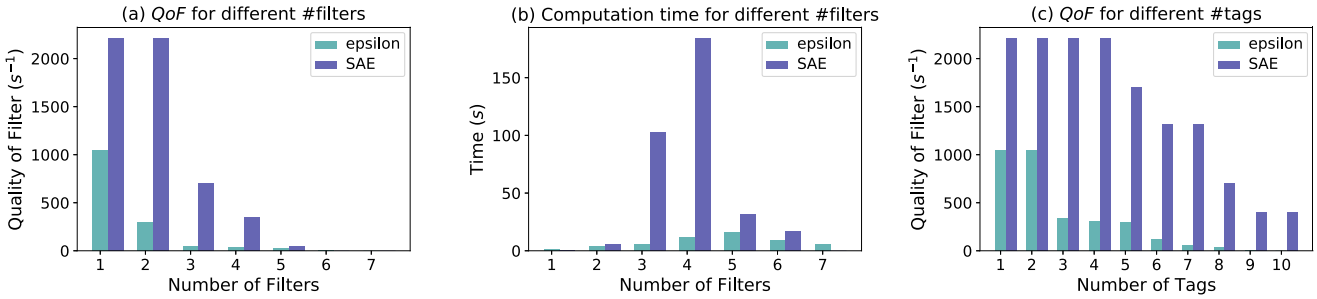

Fig. 1: Optimal *QoF* and SMT-solver timing measurements for epsilon and SAE under different numbers of filters and tags.

B. Results

For the epsilon benchmark and an unlimited number of hardware components with 1 mask and 1 associated tag, we obtain a perfect filter configuration with 7 filters within 5.64 seconds. Pölzlbauer et al. [3] report identical results without stating the runtime. As illustrated in Fig. 1(a) and (b), we obtain a *QoF* of $48s^{-1}$ for a filter configuration consisting of 3 filters within 5.24 seconds. In fact, the maximum *QoF* for the epsilon benchmark is $1180s^{-1}$. Pölzlbauer et al. [3] achieve a *QoF* of $48s^{-1}$ for this scenario as well but, again, do not provide any information about the runtime. Evidently, the *QoF* steadily improves with an increasing number of masks, i.e., filters. This observation can be made as well with respect to the SAE benchmark. Please note, that a *QoF* of $2216s^{-1}$ is achieved for the SAE benchmark, i.e., its maximum possible *QoF*, under 1 as well as under 2 masks, since the number of filters is not sufficient to block any undesired message.

In general, the runtime required to obtain a solution for the SAE benchmark is higher than for the epsilon benchmark. We also evaluated another industrial benchmark with 17 desired and 12 undesired messages, for which we achieved much shorter computation times than required for the SAE benchmark as well. Details are omitted due to a non-disclosure agreement. One important aspect affecting the runtime and leading to this observation is the ID design. In contrast to the epsilon benchmark as well as to our other benchmark, which are actively used in real-world systems, the message IDs of the SAE benchmark are created randomly. Therefore, they may have been chosen unfavorably, i.e., in a way that some IDs differ by an unnecessarily high number of bits, resulting in an increase of computation time for the SMT solver.

When using only one mask with multiple tags, a perfect filter configuration for the epsilon benchmark can be obtained as well, namely, when 10 tags are available, as pictured in Fig. 1(c). The time required to compute a solution strongly varies depending on the considered number of tags, but, however, no clear tendency is discernible.

Regarding the optimization of the number of required hardware components under a given upper bound on the *QoF*, we report the actual result as well as the computation time. With an overhead of 5% of the maximum *QoF*, an average amount of 4.2 filter components is needed with a standard deviation $\sigma = 2.79$. The solution was computed within an average of 929.4 seconds where $\sigma = 1517$ seconds. For 10% overhead, in average 5.8 filter components ($\sigma = 3.31$) and a computation time of 1478 seconds ($\sigma = 2417$ seconds) are required. Under 20% overhead, an average of 6.4 filter components ($\sigma = 1.74$) was determined within 226.19 seconds ($\sigma = 183.2$ seconds). Due to high standard deviations, it is not possible to make a statement regarding the dependence of the computation time on the amount of permitted overhead. However, since the strong deviations indubitably result from the randomly generated message IDs, we emphasize our previous conclusion that the number of filters as well as the time consumed to compute a solution strongly depend on the ID design.

VI. Conclusion

In this work, we developed an approach that allows to find good or even perfect hardware filters for broadcast buses within reasonable time. In addition, we investigated how *QoF* and computation time depend on the chosen number of filters as well as of tags. Not least, we discovered that the resulting filter configurations as well as the computation time primarily depend on the actual ID design. Hence, message IDs should be chosen systematically, taking the desired/undesired message subsets of all bus participants and the resulting filter design into account right from the beginning.

References

[1] Bosch, "Controller Area Network specification 2.0," 1991.
[2] "Road vehicles – Local Interconnect Network (LIN)," ISO Std., 2016.
[3] F. Pölzlbauer, R. I. Davis, and I. Bate, "Analysis and optimization of message acceptance filter configurations for controller area network (CAN)," in *Int. Conf. on Real-Time Networks and Systems (RTNS)*, 2017.
[4] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, Apr. 2007.
[5] A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, Feb. 2009, vol. 185.
[6] R. J. Nelson, "W. V. Quine. The problem of simplifying truth functions. The American mathematical monthly, vol. 59 (1952), pp. 521–531. (offprint 1952, on sale by the Mathematical Association of America.)," *Journal of Symbolic Logic*, vol. 18, no. 3, pp. 280–282, Sep. 1953.
[7] "Class C Application Requirement Considerations," SAE J2056/1, Standard, 1994.
[8] V. Lesi, I. Jovanov, and M. Pajic, "Network scheduling for secure cyber-physical systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2017.
[9] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.