Modeling and Verification of Starvation-Free Bitwise Arbitration Technique for Controller Area Network Using SPIN Promela

Haklin Kimm(✉) and Hanke Kimm

Computer Science Department, East Stroudsburg University of Pennsylvania, East Stroudsburg, PA 18301, USA
hkimm@esu.edu, kimm.hanke@gmail.com

Abstract. The Controller Area Network (CAN) is a message based communication service working over high-speed serial bus systems, and mostly used in the automotive industries and real-time communication systems. The CAN bus connects several independent CAN modules and allows them to communicate and work together asynchronously and/or synchronously. All nodes can simultaneously transmit data to the CAN bus, and the collision of multiple messages on the bus is resolved by a bitwise arbitration technique that operates by assigning a node with a low-priority message to switch to a "listening" mode while a node with a high-priority message remains in a "transmitting" mode. This arbitration mechanism results in the starvation problem that lower-priority messages continuously lose arbitration to higher-priority ones. This starvation is seen as a critical section problem with priority scheduling, where a synchronization technique is required at the entry and exit of the bus. The techniques of non-starvation critical section with general semaphore and barrier synchronization are applied to enable the CAN bus to proceed without starvation. In this paper, we present a formal model of a starvation-free bitwise CAN arbitration protocol applying barrier synchronization and starvation-free mutual exclusion. Based on SPIN Promela, we provide that the proposed CAN starvation-free CAN bus model works correctly.

1 Introduction

CAN stands for Controller Area Network. The CAN bus is different from other bus systems that use one master processor interfaces with other sensors or slave processors while the CAN bus creates a network of devices, each of which on the network has a microcontroller or microprocessor that interfaces with other sensors or slave processors. The CAN bus does not have a master in control of the bus. Instead, each device operates independently. Unlike an Ethernet network which is location based, the CAN bus uses a message-based network. In a location-based network each device has a unique address while in a message-based network every device

S. Lee et al. (Eds.): IMCOM 2019, AISC 935, pp. 195–210, 2019.
https://doi.org/10.1007/978-3-030-19063-7_17

on the bus receives all network traffic, and the other unique aspect to the CAN bus is that unlike an Ethernet network, there are no collisions between messages from two different devices. The CAN bus employs a non-destructive arbitration scheme to determine which device transmits on the bus. Each CAN message has an arbitration field at the beginning. In case multiple devices are transmitting messages at the same time, the device that wins arbitration will send its data while the loosing devices wait until the transmitting device finishes. A device having the most dominant field wins arbitration. On the CAN bus a logic zero is dominant while a logic one is recessive. When multiple devices send a message the first part sent is the arbitration field. As each bit is transmitted all devices simultaneously listen to the bus. If a device transmits a recessive bit and hears a dominant bit on the bus then that device has lost arbitration and ceases transmission and continues to receive the remainder of the message sent by a winning device. Because the arbitration field comes before the data fields, in the end of arbitration only one device will transmit its data [4–9].

However, the aforementioned bitwise arbitration causes the lower priority CAN messages to keep losing their transmission turns at every collision of multiple messages on the CAN bus, which places the system with starvation problem as seen in operating systems, which is seen as priority scheduling algorithm that happens to make some low-priority processes waiting indefinitely. In a crowded CAN bus system, a steady stream of higher-priority messages blocks a low-priority message from occupying the CAN bus for transmission. One of solutions to this starvation problem is aging, which is a technique that increments the priority of CAN messages that keep losing their turns for the transmission [18–21]. There have been studies on resolving starvation problem on the CAN bus. Most studies on the CAN starvation problem have been approached as priority scheduling [11–16], not as barrier synchronization or starvation-free mutual exclusion.

The previous approaches based on the scheduling algorithm do not apply the bitwise arbitration properly, where all the CAN messages enter the bus simultaneously whenever the CAN bus is free, and the messages are compared bit by bit, leaving the dominant message on the bus. In previous studies, however, the candidate messages trying to occupy the CAN bus are compared first, and the dominant message is selected and placed on the CAN bus for transmission [12,14,16], and more than two nodes are not considered for arbitration simultaneously. Here we approach this CAN starvation problem, dealing multiple messages concurrently, while applying the bitwise arbitration correctly such as with bit-by-bit comparison. All the candidate messages enter the CAN bus simultaneously, but all the recessive messages leave the bus so that the dominant message can stay on the bus alone to transmit. In this paper we present a starvation-free bitwise arbitration with aging, but not solely based on the priority scheduling that considers two CAN messages at a time as seen in previous studies [12,14–16].

The remainder of this paper is organized as follows: Sect. 2 introduces Controller Area Network with its frame format of CAN 2.0 as well as arbitration

method. Section 3 describes the previous studies related to CAN arbitration problem. Section 4 proposes and describes the starvation-free bitwise arbitration algorithm. Section 5 describes the SPIN program that has been developed to verify the proposed starvation-free algorithm, analyzes the program. Section 6 concludes the paper.

Field	SOF	Arbitration		Control			Data	CRC		ACK		EOF
	SOF	IDENTIFIER	RTR	IDE	r0	DLC	Data	C_S	C_D	A_S	A_d	EOF
Length	1	11	1	1	1	4	0...64	15	1	1	1	7
Value	0	0...2031	0	0	0	0...8	x	X	1		1	127

Fig. 1. Frame format CAN 2.0

2 Controller Area Network

The CAN is known to be very suitable to any real time systems with its low cost and high reliability as a network. Furthermore, the amount of wiring between components or modules is drastically reduced by using a shared data bus, CAN, instead of hardwired point-to-point connections. In a CAN system, CAN controllers and other components are connected over a shared CAN bus so as to avoid the need of having the components with point-to-point connections that accrue a large amount of wiring, more complex electric circuits and noise, which result in a less effective and reliable system. However, the CAN bus is not well-suited for very fast data transmission such as multimedia applications but well-suited for soft real-time systems. Also, the CAN bus system is too sophisticated and expensive for applications with low data rates, in which only few parts of the system are involved in the transmission: sun roofs or heating systems [1–3,5–9]. CAN standard frame format is shown in Fig. 1.

If two or more CAN controllers are transmitting at the same time then a dominant (‘0’) bit overrides a recessive (‘1’) bit and the value on the CAN bus will be in the dominant (‘0’) state. This mechanism is used to control access to the bus while comparing identifier fields. The CAN protocol calls for nodes to wait until a bus idle is detected before attempting to transmit again. If two or more nodes start to transmit at the same time, then by monitoring each bit on the bus, each node can determine if it is transmitting the highest priority message (with a numerically lower identifier) and should continue or if it should stop transmitting and wait for the next bus idle period before trying again. As the message identifiers are unique, a node transmitting the last bit of the identifier field, without detecting a ‘0’ bit that it did not transmit, must be transmitting the highest priority message that was ready for transmission at the start of arbitration. This node then continues to transmit the remainder of its message, all other nodes having backed off. This arbitration scheme is equivalent to logical “and” operation. CAN protocol does not use a global time to

control the bus. Instead each CAN controller uses its own clock. The CAN protocol therefore requires the nodes re-synchronize at each message transmission. Specifically, every node must synchronize with the leading edge of the start of frame bit caused by whichever node starts to transmit first in order to have bus arbitration work properly [5–9]. Four types of frames are used in the CAN protocol: Data frame, Remote frame to request the transmission of a data frame of the same frame identifier by a source, Error frame is used to destroy the frame, and Overload frame is used to provide for an extra delay between frames [1–3].

3 Previous Work

3.1 Distributed CAN System

With CAN, the functions of the distributed control systems perform well with more enhanced modularity and provide well-organized distributed controls. Under this setup the proposed distributed control system with CAN is able to perform and show the speedy synchronization moves while maintaining proper timing of the network in order to keep a flawless performance so that synchronization errors between components and robots are close to nil.

The developed program in our previous CAN work [10] provides general control of robots that communicate via CAN bus, and also presents the options to run a high-level coordinated task involving all robots with multiple moves where most movements are implemented in event-triggered control that provides a more efficient utilization of resources and a better serialization of tasks. With this, we are able to simulate the behavior of an assembly line or similar process that requires constraints on time and the need for individual activities to be performed in synchronous and/or asynchronous manners, or with/without coordinating robot moves. Our developed program works with the system commands that check the status of the distributed robots over CAN before implementing any move, since any fault or failure of a robot on our CAN system results in halting the whole CAN network and mandates a cold-start of the system again.

3.2 CAN Arbitration with Priority Scheduling

The previous CAN system had been implemented in two different ways: synchronous and asynchronous ways in order to see the accuracy and efficiency of the proposed CAN system. In synchronized move the robots of the system perform the same activities via control of the main microcontroller, which handles serial port initiation, robot arm initiation, macro job launching, interactive control via direct command, inverse kinematics calculation and control, monitoring and status querying and others. In asynchronous control, the microcontroller handles locally-connected robot I/O, and accepts the defined commands to initiate tasks. However, when implementing the system in asynchronous, we faced with unwanted situation of all the network of robots were stopped and in a deadlock situation so as to restart the whole network again. The CAN network faults of our proposed system were mostly caused by that low-priority CAN messages were dropped off because of failing to send over the CAN bus.

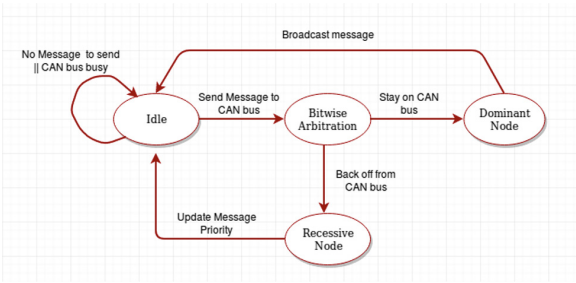

Fig. 2. State transition digram of bitwise arbitration mode

The low-priority CAN messages yielding to recessive mode are seen as a back-off protocol that has long been used to reduce contention on shared resources such as communication channels, and that has also been applied to improve the performance of shared memory algorithms in real systems [22]. One of the earlier work on resolving CAN arbitration problem related to starvation is applied by using a master node as shown in Fig. 2 [12]. The master node monitors CAN network all the time and remains silent as far as it receives messages of all the nodes but become active as it finds that some nodes are not contributing in the communication. The master node is in charge of changing the priorities of the low-priority nodes to higher priority so that the updated message can occupy the bus next round, broadcasting its message. This makes a CAN network in starvation-free mode. However in this approach, the master node should find out which low-priority messages have been backed off the CAN bus after the arbitration. It is not determinant in finding the messages that are backed off in this paper [13]. In [11], CAN arbitration model has been developed and implemented by using UPPAAL but the starvation-free CAN model has not been provided. The proposed model does not show the starvation problem as synchronization problem. In [15], the starvation-free CAN scheme has been developed and implemented using the Priority Inversion that takes 1's complement of the leftmost significant bit of a losing CAN message from the CAN arbitration. The proposed scheme works well for small number of messages in the CAN arbitration, and shows its correctness and implementation for one CAN message having lost its contention and tried the contention again after converting its leftmost bit to zero. In case, however, there are more than several messages on the bus to send their messages simultaneously, it is easily expected that there are more than one losing messages that participate in the next round of the CAN arbitration after resetting their left-most bit to zero in the proposed priority inversion scheme. Hence, more than several priority-inversion messages participate the CAN arbitration again to win the bus to send its message, so that the lowest-priority message lose to other priority-inversion messages. The priority-inversion scheme [15] and ID-rotation method [17] work well for the small number of messages and the arbitration between two messages, but do not work for large number of messages that participate again for the CAN arbitration after resetting their

left-most bit to zero. In [16], CAN with flexible data rate (CAN FD) is applied in order to reduce the traffic congestion of current controller area network (CAN). In this paper we apply studies related to the synchronization with multiple processes and shared memory [22–25].

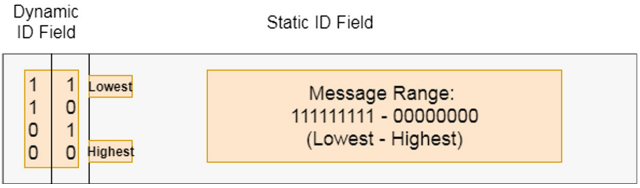

Fig. 3. Message ranges with dynamic and static IDs

In the following, we propose the CAN starvation problem as one of the synchronization problem, on which the well-known general semaphore algorithms [21,24,26] are updated to implement using SPIN Promela.

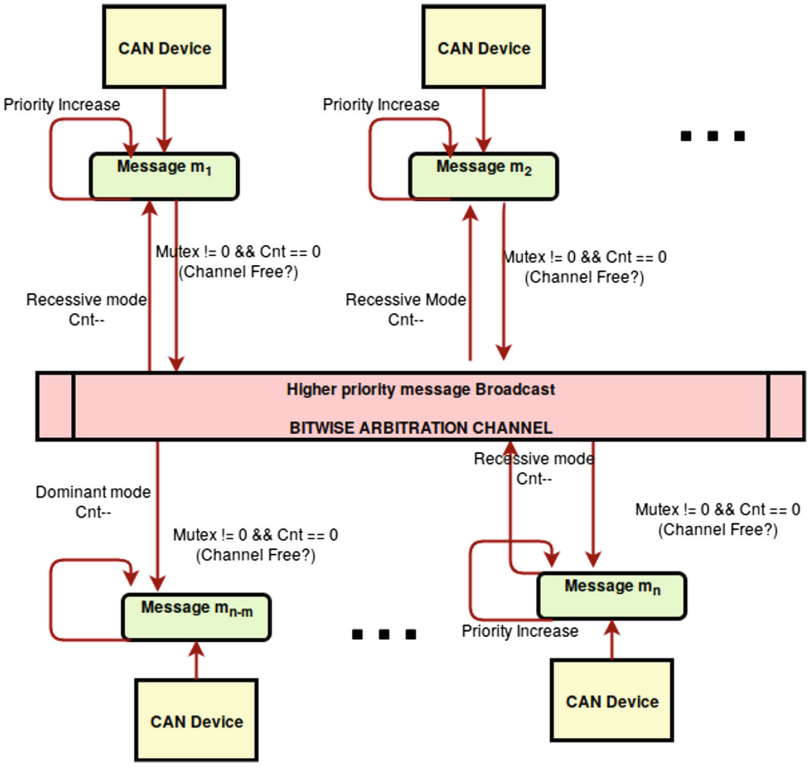

Fig. 4. Starvation-free bitwise arbitration model

4 Starvation-Free Bit-Wise Arbitration

4.1 Dynamic Message ID

The 11-bit message identifications are divided by 2-bit dynamic ID field with 9-bit static ID field as shown in Fig. 3. However, this dynamic-ID field can be flexible to be larger than 2 bits upon various network applications, and its size is lengthened more on CAN 2.0B Extended Frame where 29-bit identifiers are used. The messages using the static IDs start from the lowest number of 1023 with bit "111111111" end with the highest number of 0 with bit "000000000". With the 2-bit dynamic ID field, any recessive back-off messages can retry to occupy the CAN bus up to four times. Each 11-bit is initialed with the dynamic field ("11"), followed the static field with any number so that in the proposed system, none of the CAN messages can not use other values for its dynamic field unless it is recessive from losing in the CAN Arbitration. Each recessive message being caused by losing in the arbitration decrements its dynamic ID by one by one each round: $11 \rightarrow 10 \rightarrow 01 \rightarrow 00$. Thus each message in the proposed system is guaranteed to transmit its message in four tries.

4.2 Starvation-Free Model Correctness

As mentioned in the previous section, CAN arbitration problem, causing to starve low-priority messages but simply trying to resolve it as priority scheduling problem, does work exactly. The previous methods upon the starvation-free problem make all the messages to compare each other before allowing the dominant message occupy the CAN bus. In addition, the messages are not compared in bit by bit as CAN arbitration works, rather compared with 11-bit message identification numbers that can create up to 2048 different identifiers.

The CAN bus starvation problem has been as process scheduling in operating systems. In this paper, we consider the CAN bus as Critical Section since only one message occupy for broadcasting messages to all the CAN nodes. However, this CAN bus should open to any CAN nodes trying to send their messages simultaneously if the CAN bus is free at the time the CAN nodes check to see it open. This simultaneous access to the CAN bus leads the CAN Critical Section problem in the following changes.

- Property 1: Multiple nodes can access the CAN bus, Critical Section simultaneously.
- Property 2: Only one node stay on the Critical Section to enable for the transmission.
- Property 3: The CAN messages on the Critical Section run by bit-by-bit comparison.
- Property 4: Low-priority messages retire from the Critical Section if not dominant.
- Property 5: The identification number of the retired low-priority messages are decremented.

– Property 6: Any low-priority messages trying to access the CAN bus eventually enable to transmit their messages.

In operating systems [18–20], the critical section problem is defined such as no other process is allowed to run in its critical section when one process is running in its critical section. The critical section is considered a segment of code in which the process may be changing shared variables, writing to a file, and so on. A solution to the critical section problem demands to satisfy the following three conditions: Mutual exclusion, Progress, and Bounded Waiting.

The proposed properties are discussed to see whether to satisfy the three critical-section requirements. The first requirement of Mutual Exclusion says that if one process is running in its critical section, then no other processes are not allowed in their critical sections. The properties 1 and 2 enable the proposed algorithm to satisfy the Mutual Exclusion, but the CAN bus allows multiple messages to enter the critical section in which no other message runs for the transmission. Only one winning highest-priority message runs in the critical section, meaning that the winning on the CAN bus can transmit its own messages. The second requirement of Progress says that if the critical section is free and some processes wish to enter the critical section, then this process wanting to enter the critical section should not be delayed indefinitely. Only those processes not running in their remainder section can participate in deciding which one will enter the critical section. The properties 3 and 4 enable the Starvation-Free algorithm to satisfy the Progress. As mentioned above, only the messages in the critical section, say CAN bus, decide which message to keep staying and running in the critical section by making the lower-priority messages yield the CAN bus to higher-priority message. Finally only one node on the CAN bus running to broadcast its own message. This satisfy the second requirement that the nodes on the CAN bus decide which node stays in the critical section to run since these nodes are not in their remainder sections. The third requirement of Bounded Waiting says that no process should have to wait forever to enter the critical section if the process has made a request to enter. The properties 5 and 6 allow the algorithm to suffice the Bounded Waiting because any low-priority messages wishing to enter the CAN bus for transmitting their own messages

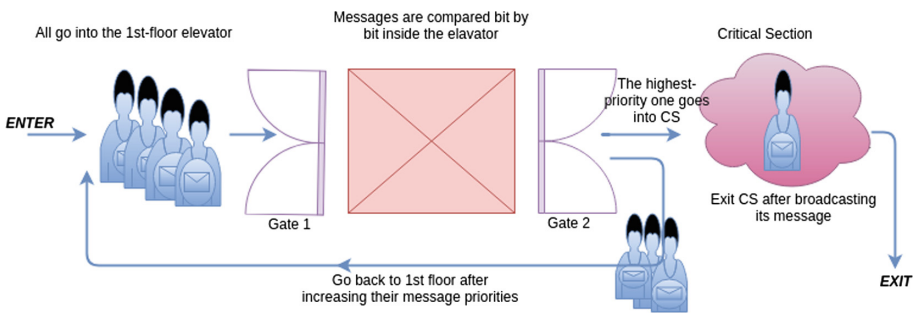

Fig. 5. Starvation-free mutual exclusion model

eventually enable to occupy the CAN bus because the property 5 allows the retired nodes to decrement its identification number of their corresponding fields, ended up increasing their message priorities.

4.3 Starvation-Free Mutual Exclusion with Semaphores

There are studies of critical section problem without starvation using weak semaphores. In weak-semaphore model, if a process performs a Wait operation and another process is waiting on that semaphore, then the first process is not allowed to immediately perform another Signal operation on that semaphore. Instead, one of the waiting processes must be given its turn. Hence, a single weak-semaphore would be in starvation situation. However, using few of them can guarantee starvation-free mutual exclusion [27]. In [26] Starvation-free critical section algorithm, shown in Algorithm 1, works with weak and split-binary semaphores. A system of semaphores $s_1+s_2+...+s_n$ forms a so-called split-binary semaphore if system preserves the invariant: $s_1+s_2+...+s_n+\#\{q \mid q\ in\ PV\} = 1$, where PV is a set of locations and q is a process [25].

The starvation-free algorithm of [26] is applied to explain Fig. 5. As seen in [26], each elevator door maps to a gate such as Gate1 to a elevator door on the first floor and Gate2 for the second floor. There will be no queue for waiting processes. This algorithm works with a batch of arrivals, which start to walk into the first-floor elevator and the door is closed when no more processes arrive. This Algorithm guarantees starvation-free mutual exclusion since the first elevator door is open again only after a whole set of processes in the second-floor elevator has been served.

4.4 Bit-Wise Arbitration Model Applying Non-starvation Mutual Exclusion with General Semaphores

The Starvation-Free Bitwise Arbitration can be seen as a synchronization problem such as starvation-free mutual exclusion shown in [26]. As seen in Fig. 5, our proposed starvation-free bitwise model can be explained as an elevator problem. CAN messages are arrived almost at the same time to enter an elevator on the first floor. When there are no more arriving CAN messages, the elevator goes to the second floor and lets the dominant highest-priority message in the elevator walks into CS, *Critical Section*. CAN bitwise arbitration is done inside the elevator while going from first floor to second. The remaining messages in the elevator, who could not be in CS, go down to the first floor by the stairs to enter the first-floor elevator again with other arriving new messages. While walking down the stairs, the priorities of remaining recessive messages are incremented.

Algorithm 1. Udding's Starvation-free Algorithm

```
1  byte gate1 = 1, gate2 = 0; //semaphores
2  byte cntGate1 = 0, cntGate2 = 0 ;
3  loop forever
4      Non Critical Section
5      wait(gate1);
6      cntGate1++;
7      signal(gate1); //First gate
8      cntGate2++;
9      cntGate1−−;
10     if (cntGate1 > 0) →
11         signal(gate1)
12     else signal(gate2)
13     wait(gate2); //Second gate
14     cntGate2−−;
15     Critical Section
16     if (cntGate2 > 0) →
17         signal(gate2);
18     else signal(gate1);
```

Algorithm 2. CAN Arbitration with Starvation-free Mutual Exclusion with General Semaphores

```
1  byte flagCAN = 1, busCAN = 1; //semaphores
2  byte cnt = 0, ent = #_of_processes;
3  loop forever
4      wait(flagCAN); //simulated wait
5      wait(busCAN);
6      cnt++;
7      if (cnt < ent) →
8          signal(flagCAN)
9      signal(busCAN);
10     Critical Section (CAN Bus Arbitration)
11     wait(busCAN); //simulated signal
12     cnt−−;
13     if (cnt <= 0) →
14         signal(flagCAN);
15     else
16         wait(flagCAN);
17     signal(busCAN);
```

In the proposed Bit-wise Arbitration method shown in Algorithm 2, the CAN node with its message are defined as a process in lieu of the simplicity of describing the proposed synchronization problem. The processes in the algorithm share the following data structures: semaphore mutex, and int count. The semaphore mutex is initialized to 1, and count to 0. The mutex is used to ensure mutual exclusion of using CAN bus, the variable *count* is used to keep track of how may processes are currently on the bus before processing the Bit-wise Arbitration. The variable count is used by each lower-priority process yielding the CAN bus, say Critical Section, to higher-priority process. When the processes enter the first-floor simultaneously, each process increments the variable *count* by one; when the yielding processes exit the second-floor elevator, the variable *count* decrements by one as seen in Fig. 5. However, the semaphore mutex is changed to 0 by the first process that enters the critical section even though other processes enter the elevator simultaneously. Any process that enters the CAN bus do not change the mutex value if the variable count is greater than 0. Other processes outside the critical section have to wait if the semaphore mutex is 0. This kind of situation is seen in the reader-writer problem, where the first reader entering the critical section can change the semaphore, *write* to 0 and the last reader leaving the critical section can change the semaphore *write* to 1.

The proposed starvation-free technique shown in algorithm 2 provides a solution of the critical section problem that allows multiple processes, say CAN messages, in the critical section simultaneously but only the highest-priority message occupies the bus for broadcasting. The variable *ent* is initialized with the maximum number of processes to transmit messages on the CAN bus simultaneously. The Algorithm 2 starts in the following way. Since the semaphore busCAN is initialized to 1, the first CAN message attempting to enter a *simulatedwait* statements will succeed in executing s4: wait(busCAN). This process will promptly increment *cnt*, and implement a sequence of *simulatedwait* operations. This causes *cnt* to have a positive value. The first process and each subsequent processes, up to a total number of *ent*, are allowed to carry out s4 through s10. All the process are in CR simultaneously, where only the winner broadcasts its own message in CAN bus system. In exiting CR all the recessive messages including the winner in CR enter a *simulatedsignal* statements so that succeed in executing s11: wait(busCAN). This process will promptly decrement *cnt*, and a sequence of simulated signal operations from s12 thru s17. The last message in the *simulatedsignal* will cause *cnt* to have a zero so as to implement s14: signal(flagCAN). Then CAN bus is ready for the next round of bus arbitration again. This is shown clearly in Fig. 5.

In the proof, we will reduce the number of steps in the induction to three. The binary semaphore busCAN prevents the statements s5..9 from interleaving with the statements s11..17. S4 can interleave with statements s5..9 or s11..17, but cannot affect their execution, so the effect is the same as if all the statements s5..9 or s11..17 were executed before s4. The similar proof is shown well in [21,24,25]. The conjunction of the following formulas is invariant with the same notations, and the inductive proof of this general semaphore algorithm is also mentioned well in [21].

1. entering $\rightarrow$ (flagCAN = 0),
2. entering $\rightarrow$ (cnt >= 0),
3. #entering $\leq$ 1,
4. (flagCAN = 0) $\wedge$ ($\neg$ entering) $\rightarrow$ (cnt = 0),
5. (cnt $\geq$ ent) $\rightarrow$ (flagCAN = 1),
6. (flagCAN = 0) $\vee$ (flagCAN =1)

5 Starvation-Free Model Using SPIN

The proposed starvation-free algorithm provides a solution of the critical section problem that allows multiple processes, CAN messages, in the critical section, and its implementation is attached below. We illustrate this outcome with the SPIN Promela program shown in Algorithm 3.

```
stc369@stc369:/CAN-PROMELA/spin can5.pml
MSC: process 193 entered CAN bus
             with message 66 lowPri 193.
MSC: process 198 entered CAN bus
             with message 68 lowPri 193.
MSC: process 197 entered CAN bus
             with message 67 lowPri 193.
MSC: process 192 entered CAN bus
             with message 65 lowPri 192.
Winner pid 192 broadcasts its message 65
Recessive node with pid 194, message 66
Recessive node with pid 198, message 68
Recessive node with pid 197, message 67

MSC: process 198 entered CAN bus
             with message 69 lowPri 198.
MSC: process 132 entered CAN bus
             with message 67 lowPri 132.
MSC: process 129 entered CAN bus
             with message 66 lowPri 129.
MSC: process 196 entered CAN bus
             with message 65 lowPri 129.
Winner pid 129 broadcasts its message 66
Recessive node with pid 198, message 69
Recessive node with pid 196, message 65
Recessive node with pid 132, message 67
```

As seen in Algorithm 3, the variable *ent* is initialized with the maximum number of processes that can be in the critical section simultaneously. In this example *ent* is initialized with 4, so that at most 4 CAN messages are in for arbitration simultaneously. Statements s10..12 implement *signal(flag)* allow multiple

Algorithm 3. Implementation of Starvation-free Model

```
byte flag = 1, lock = 1;   //semaphores
byte cnt = 0;
byte ent = 4; //# of CAN messages for arbitration in this example
byte lowMsg, lowPri=255;
inline SendMSG() {
        wait(flag);
        wait(lock);
        atomic { //each message for CAN arbitration
            cnt++;
            if
            :: (cnt < ent) →
              signal(flag);
            if
            :: (lowPri >= id) →
              lowMsg = msg; lowPri = id;
            :: else
            fi
        }
       signal(lock);

       if
        :: (cnt == ent) →
              the winner broadcasts in CR
              lowMsg, lowPri = 255; //reset lowPri and lowMsg for next round;

        wait(lock);
        cnt−−;
       if (cnt <= 0) →
         signal(flag);
       else
         wait(flag);
       signal(lock);
}
proctype NodeCAN(byte id; byte msg) {
         SendMSG();
}
init {
        atomic {
           run NodeCAN(0,65); run NodeCAN(1,66);
           run NodeCAN(2,67); run NodeCAN(3,68);
            }
       /*updated messages with the higher priority
           join the CAN bus again */
}
```

processes to enter the critical section that is immediately followed by the simulated arbitration for CAN bus shown in statements s20..23. Statements s24..30 simulate the *signal(flag)* operation that allow CAN bus to be available for the next round of message transmission.

As seen above, CAN messages have been created with 8 bits such that leftmost 2 bits are filled with 1 in order to provide dynamic priority, where message IDs are from 11000000 (=192) to 11111111 (=255), and dynamic IDs are from 11,10,01,00. As seen in the first round of CAN messages in Algorithm 3, the message 192 has been selected as winner from 4 messages that entered the CAN bus simultaneously, and the remaining IDs retired as recessive node. The 2 recessive nodes participated again in the second round of CAN bit arbitration: 129 (10000001) from 193 (11000001), and 132 (10000100) from 196 (11000100) after applying dynamic bit arbitration technique that replaces the leftmost two bits for the next round. Then this SPIN implementation enables low-priority IDs to transmit their messages.

6 Conclusion

In this paper we presented a new starvation-free model for CAN arbitration, where the low-priority messages yield the CAN bus to the higher-priority messages so that the low-priority keeps losing the arbitration not being able to transmit its message on time. Previous works [11–17] on this starvation problem were trying to resolve it as priority scheduling problem in which the CAN bus arbitration is executed just before entering the CAN bus so as to make one message access the CAN bus. The CAN bus arbitration allows all the messages to join the bus simultaneously and then selects the highest-priority message as dominant node that occupies the bus to broadcast its message to all the recessive nodes on the CAN network. Hence, this problem is interpreted as a non-starvation critical section model [25,26] not as priority scheduling. In our proposed model, we consider the CAN bus as critical section where multiple messages can access simultaneously for the arbitration, so that the multiple messages should access the critical section simultaneously once the CAN bus is free. This problem can be seen as a starvation-free critical section model where multiple processes are allowed to enter the critical section for the CAN bus arbitration. The well-known barrier technique and non-starvation critical section model applying general semaphore, that allow multiple process to join the critical section, have been applied to our proposed model, and its correctness has been provided. Furthermore, the proposed algorithm has been implemented by using SPIN Promela [28–30] to provide the model's correctness. The simulation of the proposed model using SPIN reveals that our starvation-free CAN model works well. At this time we are working further on a streamlined approach that the proposed model can be applied in other areas with ease.

References

1. Albert, A., Bosch, R.: Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. Proc. Embed. World, 235–252 (2004)
2. CAN specification version 2.0. Robert Bosch GmbH, Stuttgart, Germany (1991)
3. Gil, J.A., et al.: A CAN architecture for an intelligent mobile robot. In: Proceedings of SICICA-97, pp. 65–70 (1997)
4. Fuhrer, T., et al.: Time-triggered Communication on CAN (Time-triggered CANTTCAN). In: Proceedings of ICC 2000, The Netherlands, Amsterdam (2000)
5. Hartwich F., et al.: Timing in the TTCAN network. In: Proceedings of 8th International CAN Conference, Las Vegas (2002)
6. Homepage of the organization CAN in Automation (CiA) (2004). http://www.can-cia.de
7. Rett, J.: Using the CANbus toolset software and the SELECONTROL MAS automation system, Control System Center, University of Sanderland (2001)
8. Magnenat S., et al.: ASEBA, an event-based middleware for distributed robot control. In: International Conference on Intelligent Robots and Systems (2007)
9. Davis, R.I., et al.: Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Syst. **35**, 239–272 (2007)
10. Kimm, H., Kang, J.: Implementation of networked robot control system over controller area network. In: Proceedings of the Ninth IEEE International Conference on Ubiquitous Robots and Ambient Intelligence, Daejeon, Korea, 26–29 November 2012 (2012)
11. Pan, C., et al.: Modeling and verification of CAN bus with application Layer using UPPAL. Electron. Notes Theor. Comput. Sci. **309**, 31–49 (2014)
12. Murtaza, A., Khan, Z.: Starvation free controller area network using master node. In: Proceedings of IEEE 2nd International Conference on Electrical Engineering, Lahore, Pakistan, 25–26 March 2008 (2008)
13. MSC8122: Avoiding Arbitration Deadlock During Instruction Fetch, FreeScale Semiconductor, INC. (2008)
14. Lee, K, et al.: Starvation Prevention Scheme for a Fixed Priority Pci−Express Arbiter with Grant Counters Using Arbitration Pools. US Patent Application Publication, 14 January 2009
15. Lin, C.-M.: Analysis and modeling of a priority inversion scheme for starvation free controller area networks. IEICE Trans. Inf. Syst. **93–D**(6), 1504–1511 (2010)
16. Zago, G., de Freitas, E.: A quantitative performance study on CAN and CAN FD vehicular networks. IEEE Trans. Ind. Electron. **65**(5), 4413–4422 (2018)
17. Park, P., et al.: Performance evaluation of a method to improve fairness in in-vehicle non-destructive arbitration using ID rotation. KSII Trans. Internet Inf. Syst. **11**(10) (2017)
18. Silberschatz, A., et al.: Operating System Concepts, 9th edn. Wiley, Hoboken (2016)
19. Tanenbaum, A., Bos, H.: Modern Operating Systems, 4th edn. Pearson Prentice-Hall, Upper Saddle River (2015)
20. Stallings, W.: Operating Systems: Internals and Design Principles, 9th edn. Pearson Prentice-Hall, Upper Saddle River (2016)
21. Ben-Ari, M.: Principles of Concurrent and Distributed Programming, 2nd edn. Addison-Wesley, Boston (2006)
22. Ben-David, N., Belloch, G.: Analyzing contention and backoff in asynchronous share memory. In: Proceedings of PODC 2017, Washington, DC, USA, 25–27 July 2017 (2017)

23. Dalessandro, L., et al.: Transcational mutex locks. In: Proceeding of European Conference on Parallel Processing, Italy, 31 August–3 September 2010 (2010)
24. Trono, J., Taylor, W.: Further comments on "a correct and unrestrictive implementation of general semaphores". ACM SCIGOPS Oper. Syst. Rev. **34**(3), 5–10 (2000)
25. Hesselink, W.H., IJbema, M.: Starvation-free mutual exclusion with semaphores. Form. Asp. Comput. **25**(6), 947–969 (2013)
26. Udding, J.: Absence of individual starvation using weak semaphores. Inf. Process. Lett. **23**, 159–162 (1986)
27. Friedberg, S.A., Peterson, G.L.: An efficient solution to the mutual exclusion problem using weak semaphores. Inf. Process. Lett. **25**, 343–347 (1987)
28. Wikipedia page. https://en.wikipedia.org/wiki/Promela. Accessed 25 Oct 2017
29. Gerth, R.: Concise Promela reference (1997). http://spinroot.com/spin/Man/Quick.html. Accessed 25 Oct 2017
30. Ahrendt, W.: Lecture Slides: Introduction to Promela - in the course Software Engineering using Formal Methods, Chalmers University (2014). cse.chalmers.se/edu/year/2014/course/.../PROMELAIntroductionPS.pdf. Accessed 25 Oct 2017