

Analysis of Real-Time Backplane Bus Network Based on Write Posting

Minyoung Sung, Taehyoun Kim, Naehyuck Chang and Heonshik Shin
Department of Computer Engineering
Seoul National University, Seoul 151-742, Korea
{mysung, thkim, naehyuck, shinhs}@comp.snu.ac.kr
Phone: +82-2-880-7295, Fax: +82-2-874-3104

Abstract

The backplane bus network protocol is a mechanism to facilitate the standard networking on the backplane bus. Usually a link-level network protocol is designed and implemented to provide the bus networking. In this paper we introduce a scheduling analysis for real-time communication in the backplane network. Unlike traditional communication analysis, we must consider the write posting, a feature of bus interface which improves the overall schedulability dramatically. For an exact real-time analysis, a methodology is also developed to model the timing behavior of the bus.

1 Introduction

Recent advances in computing and communication technologies have made it possible to design and implement a variety of multimedia applications such as a video-on-demand, a video conference and a multimedia electronic mail, which in turn require more powerful systems with good processing powers and high-speed networks. Together with sophisticated real-time technologies, a multi-processor system with low-price processors interconnected by *backplane bus* can be a cost-effective solution.

The backplane bus provides relatively high data transfer rates and is effectively utilized by tasks transferring large volumes of data. Applications often access the shared memory regions directly with the bus protocol instead of high level standard message-based protocols. In recent days, however, as the performance of embedded systems increases, it is getting more popular to use the standard network protocol even for the communication between processing modules in the same rack. The benefits of using the backplane network protocols are the short program development cycles and the reduced software debugging effort.

With all the advantages of the backplane network protocols, however, one will be reluctant to use it as a means

for real-time communication without a robust analysis of its timing behavior. Most previous works on real-time communication have been based on an ideal bus for the bounded access delay. A bus scheduling based on rate monotonic analysis has been introduced by [5]. It describes an integrated processor and bus scheduling. Recent work [6] has addressed the scheduling of messages on a timed token passing bus and a priority preemptive real-time bus. Each of these papers used an idealized bus scheduling or made certain assumptions of the ideal bus behavior. More recently, there have been several papers reporting the issues in commercial buses. [3] derives an idealized scheduling analysis for the Controller Area Network (CAN) bus and describes the implementation issues. For some common system buses such as VMEbus, PCI and Futurebus, [2] models in detail the physical operation of the buses.

In this paper we choose the *BusNet*, a draft backplane network protocol and introduce a real-time scheduling analysis for the BusNet. We describe architectural features and a modeling methodology for real-time bus transfers. The analysis is carried out considering the Logical Link Control (LLC) and Media Access Control (MAC) protocols. A schedulability analysis method is developed as well for real-time tasks communicating over the BusNet. Based on the worst-case response time analysis by [1], the analysis is designed taking the characteristics of bus interface into account. We apply the analysis for the BusNet over the VMEbus[7] since the BusNet specification uses only the basic features common in most backplane hardware and the VMEbus is one of the most widely used backplane. Thus our method is expected to be applicable to other backplane network protocols without further effort.

The rest of this paper is organized as follows. Section 2 describes the backplane bus network model. In Section 3, we present a scheme for bus modeling and develop a schedulability analysis. This paper ends with concluding remarks and future works in Section 4.

2 BusNet: A Backplane Bus Network Model

A system model using BusNet for interprocessor communication is shown in Figure 1. The system is composed of a number of processing nodes (boards). Each node or participant is assumed to include a master module as well as a slave module.

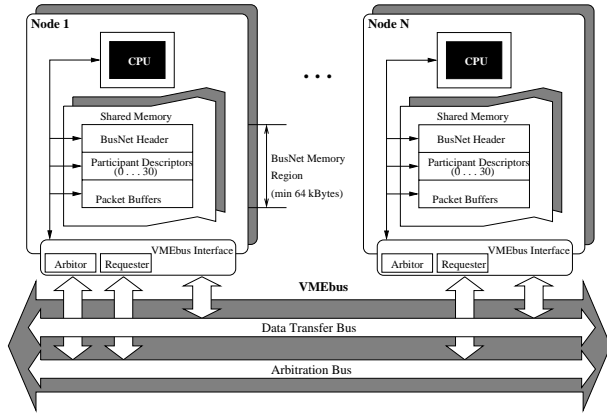


Figure 1. BusNet backplane network model

As shown in Figure 1, the *interface module* controls the use of VMEbus, and consists of a *requester* and an *arbiter*[7]. The *arbitration bus* is comprised of *bus request lines* and *bus grant lines*. The VMEbus supports four levels of priorities for the bus request. While the same priority bus requests from all the requesters are connected together in the form of a wired-OR, the bus grant lines are daisy-chained across the participants and used for the arbiter to award use of the bus. For the simplicity of description, *bus* refers to the data transfer bus hereafter.

Three types of arbiters are described in the VMEbus standard[7]: prioritized (PRI), round-robin (RRS) and single level (SGL). In this paper, PRI arbiter will be considered. A PRI arbiter issues a bus grant corresponding to the highest priority among bus requests detected. When a requester detects a bus grant signal but did not requested it, it passes the signal down to the next participant on the daisy-chain. If the requester has requested the bus, it drives the busy signal active. The daisy-chain greatly reduces resource complexity and usually works well under light traffic conditions. Its behavior, however, makes it difficult to be used for real-time systems. For instance, the participant located at the end of the daisy-chain can be left starved under heavy traffic. For the guaranteed access, the requesters must be configured in a FAIR mode. The FAIR requester, after being granted access to the bus, does not issue another request if there are other bus requests pending on its own priority level. Thus this paper uses the FAIR requesters.

Table 1. Steps for a packet transmission

step	Description	bytes
1	T checks to see if R is ready to receive packets. $map_t.PD[r].receive_status == RDY$	
2	T determines the buffer offset and size parameters. $packet_buff_offset = map_t.PD[r].buff_offset$ $packet_buff_size = map_t.PD[r].buff_size$	
3	T sets Receive Status to IDLE. $map_t.PD[r].receive_status = IDLE$	
4	T determines the VME address of R's packet buffer.	
5	T transfers a packet to R's packet buffer.	2048
6	T updates the packet's Sequence Number in R' PD. $map_r.PD[t].sequence_number + = 1$	4
7	T sets Transmit Status to RDY. $map_r.PD[t].transmit_status = RDY$	1
8	T generates a mailbox interrupt to R.	1
9	R checks the Sequence Number. $map_r.PD[t].sequence_number == next_val$	
10	R sets Transmit Status to IDLE. $map_r.PD[t].transmit_status = IDLE$	
11	R stores the address of newly allocated packet buffer. $map_t.PD[r].buff_offset = offset_of_new_buff$	4
12	R sets Receive Status to RDY. $map_t.PD[r].receive_status = RDY$	1
13	R generates a mailbox interrupt to T.	1

BusNet, originally developed by Force Computers Inc., is a proposal for an open and standard protocol[8, 9, 10]. It allows multiple VME CPU-boards and intelligent controller boards to communicate with each other over a standard network protocols on the VME backplane. BusNet has been implemented on a variety of operating systems to support standard network protocols such as Ethernet on the VMEbus backplane. It can be considered as a substitution for layers 1 and 2 of the ISO/OSI model. In BusNet, each participant is identified by a logical address of range between 0 and 30. It must be capable of sharing a segment of its local memory, as a VMEbus slave, for access by other participants. The shared memory region holds data structures in an area referred to as the *BusNet region* (Figure 1). Each BusNet region is composed of a BusNet header, participant descriptors, and packet buffers. The protocol descriptors (PD) in each participant descriptor contain flags and pointers to facilitate the exchange of BusNet packets between two peers. Packet transmissions in BusNet are performed by taking the steps illustrated in Table 1. In the table, T and R represent the transmitter and the receiver, respectively.

3 Scheduling Model and Analysis

3.1 Analysis of Packet Transmissions

The basic unit of data transfer in bus model is called a *transaction*. A transaction is defined as the lowest level transfer mechanism and represents one transfer on the bus. Transactions can be performed in one of several modes. We

consider two modes: default transfer (DFT) mode and block transfer (BLT) mode. In DFT mode each transaction requires a unique arbitration followed by address/data cycle. The BLT mode differs from the DFT mode in that once a master obtains the bus, it requires only one address/data cycle followed by a number of data-only cycles. Usually the address is incremented automatically. The advantage of this scheme is that transaction overhead is amortized across multiple bus transfers. The time required for a transaction in DFT and BLT mode can be defined as

$$\begin{aligned} t^{unit_dft} &= t^{arb} + t^{transf_dft} + t^{rel_dft} \\ t^{unit_blt} &= t^{arb} + t^{transf_dft} \\ &\quad + (s - 1) \cdot t^{transf_blt} + t^{rel_blt}. \end{aligned}$$

The scale factor, s , denotes the number of data cycles involved in a block transfer and depends on the adapters of concern. Notations used for modeling the backplane bus is illustrated with their typical values¹ in Table 2.

Table 2. Notations used for bus model

Notation	Description	Value
t^{unit_dft} , t^{unit_blt}	Time required for a transfer in DFT and BLT mode.	306 ns 9665 ns
t^{arb}	Time required to perform arbitration.	78 ns
t^{transf_dft}	Time required for an address/data cycle.	159 ns
t^{transf_blt}	Time required for a data-only cycle.	149 ns
t^{rel_dft} , t^{rel_blt}	Time to release the bus after completing the last data cycle.	69 ns 41 ns
s	Block transfer scale.	64
b	Width of the bus in bytes.	4 bytes

Now, we analyze the time spent on the bus to transmit a packet. Because steps 1~4 in Table 1 are executed in the local processor, we do not take these delays into account. Bus transfers start at step 5 where the time taken to transfer 2048 bytes of data is $\lceil \frac{2048}{b \cdot s} \rceil \cdot t^{unit_blt}$. Through steps 6~8, three DFT mode transactions must be performed adding $3 \cdot t^{unit_dft}$ to the total transfer time. In addition, three DFT mode transactions are required for steps 11~13. Hence assuming there is no contention for the bus, the time taken to transmit a packet equals to

$$\rho = \left\lceil \frac{2048}{b \cdot s} \right\rceil \cdot t^{unit_blt} + 6 \cdot t^{unit_dft}. \quad (1)$$

In this paper, all the real-time communications are assumed to be conducted with the highest bus priority. For each node the transfer requests of the same priority is scheduled in a round-robin order by unit transfer. If each processor p ($1 \leq$

¹Values are based on the SCV64 VMEbus interface chip by Tundra Semiconductor Corp.

$p \leq N$) has L_p packets to transmit, then the worst-case time taken for p to transmit all packets will be

$$\rho \cdot \sum_{i=1}^N \min(L_p, L_i).$$

3.2 Real-Time Scheduling Based on Write Posting

In order to guarantee that the timing requirements of all tasks are met, the communication delay must be bounded. For the communication delay, we focus on the worst case time taken between the arrival of the sender task and the reception of messages by the communication device at the destination node. For an accurate end-to-end communication delay, the delivery delay must also be considered. The delivery delay is defined as the time taken to process the message at the destination processor before finally delivering it to the destination task. The sporadically periodic task analysis can be used to obtain both a measure of the time taken for this processing and the impact of the costs on tasks on the destination processor[6]. In this paper the delivery delay is not dealt with in detail.

A task which sends messages to other tasks on a different processing node can be modeled as follows.

$$\tau_i (T_i, C_i, D_i, n_i)$$

Task τ_i has a period T_i , a deadline D_i and the worst-case computation time C_i . The task has message m to be sent where n_i denotes the worst-case number of packets contained in the message. The protocol processing overheads must be incorporated in C_i including the interrupt handling for packet transmissions. The worst-case response time of τ_i is defined as that of message m and is denoted by R_i . Then τ_i is schedulable if it satisfies the constraint, $R_i = R_i^m \leq D_i$ where R_i^m is the worst-case response time of message m .

To find the worst-case response time of the message, we must first determine if the *write posting* or *cycle decoupling* facility is supported by the VMEbus interface. Data transfer between the local bus and the VMEbus normally involves coupling handshake protocols. That is, the bus that originated the data transfer is only released when the last bus in data transfer chain (master local bus, VMEbus, and slave local bus) completes its cycle. For the decoupled cycle, however, when a cycle arrives from one bus (local or VME), it is queued in an internal FIFO of the interface controller and the originating bus is notified of cycle completion. While the interface acquires the other bus and completes the data transfer, the original bus is released and is free to conduct other cycles.

If there exists enough FIFO buffer for all bus transfer requests required to transmit a packet, the analysis can be

proceeded in the same way as in the system with a network adapter. Typically, the output routine for the network interface queues the packet on its send queue and primes an interrupt-driven routine to transmit the packet. The transmission of the packet can then be performed in a way independent of the current CPU job. To find the worst-case response time of the message m we find the release jitter of m and then add the time taken to transmit n_i packets[6]. The release jitter of the message can be thought of as the difference between the earliest and the latest releases and is bounded by R_i^c , the worst-case response time of the computation C_i . The R_i^c is defined by

$$R_i^c = \max_{q=0,1,2,\dots} (w_{i,q}^c - qT_i), \quad (2)$$

where $w_{i,q}^c$ is the width of the *level i busy period* [4].

$$w_{i,q}^c = (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_{i,q}^c}{T_j} \right\rceil \cdot C_j. \quad (3)$$

We define P as the set of all processors and $T(p)$ as the set of all tasks on processor p . Assuming task τ_i is placed on processor p , the worst-case response time of message m is given by

$$R_i^m = \max_{q=0,1,2,\dots} (R_i^c + w_{i,q}^m - qT_i), \quad (4)$$

where width of *bus busy period*, $w_{i,q}^m$ is given by

$$\begin{aligned} w_{i,q}^m &= B_i^m + l_1(\rho + \nu) + l_2(\rho + \nu) \\ &+ \rho \cdot \sum_{\forall q \in P, q \neq p} \min \left(l_1 + l_2, \sum_{\forall k \in T(q)} \left\lceil \frac{w_{i,q}^m}{T_k} \right\rceil n_k \right), \\ \text{where } l_1 &= n_i(q+1), \quad l_2 = \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q}^m}{T_j} \right\rceil n_j. \end{aligned} \quad (5)$$

In Equation (5), ρ is the time taken for the bus to transfer all data in a packet and ν is the worst-case computation time taken for the destination processor to handle packet arrival interrupt associated with steps 9 to 13 in Table 1. The processor p has $l_1 + l_2$ packets for the busy period where l_1 and l_2 respectively represent the number of packets for τ_i and all tasks of higher priority than τ_i . The term B_i^m is the blocking factor for message m . Since the transmission of packet cannot be preempted once it has begun, B_i^m equals to $\rho|P| + \nu$ where $|P|$ denotes the number of processors in set P . The last term in Equation (5) is the interference from other processors when using the bus.

If either the write posting is not supported or the FIFO capacity is insufficient even when supported, packet transmissions are performed coupled with local CPU cycle. In the worst-case, the interference from other processors must

be incorporated in $w_{i,q}^m$ also for the time taken to complete the computation of task τ_i and tasks of higher priority.

$$R_i^m = \max_{q=0,1,2,\dots} (w_{i,q}^m - qT_i). \quad (6)$$

$$\begin{aligned} w_{i,q}^m &= B_i^m + (q+1)C_i + l_1(\rho + \nu) \\ &+ \sum_{\forall j \in hp(i)} \left\lceil \frac{w_{i,q}^m}{T_j} \right\rceil \cdot C_j + l_2(\rho + \nu) \\ &+ \rho \cdot \sum_{\forall q \in P, q \neq p} \min \left(l_1 + l_2, \sum_{\forall k \in T(q)} \left\lceil \frac{w_{i,q}^m}{T_k} \right\rceil n_k \right). \end{aligned} \quad (7)$$

4 Conclusion and Future Work

In this paper, we have introduced a backplane network model and developed a modeling methodology for real-time communication. A real-time analysis was performed taking into account the effects of write posting on the schedulability.

To validate the efficiency and correctness of our analysis, we plan to carry out an extensive experiment. Also we intend to parameterize the impacts of write posting on the CPU scheduling and to quantize its cost.

References

- [1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, Sept. 1993.
- [2] K. A. Kettler, J. P. Lehoczky and J. K. Strosnider. Modeling Bus Scheduling Policies for Real-time Systems. In *Proceedings of Real-Time Systems Symposium*, pages 242–253, Dec. 1995.
- [3] K. Tindell, H. Hansson and A. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of Real-Time Systems Symposium*, pages 259–263, 1994.
- [4] J. P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proceedings of Real-Time Systems Symposium*, pages 201–209, 1990.
- [5] L. Sha, J. Lehoczky, and R. Rajkumar. Solutions for Some Practical Problems in Prioritized Preemptive Scheduling. In *Proceedings of Real-Time Systems Symposium*, pages 181–191, 1986.
- [6] K. Tindell, A. Burns, and J. Wellings. Analysis of Hard Real-Time Communications. *Real Time Systems*, 9:147–171, 1995.
- [7] VMEbus Specification: Draft Specification Revision D1.6. IEEE, May 1993.
- [8] Summary and introduction to the BusNet standard. VITA 19.0-1997, June 1997.
- [9] BusNet Media Access Control(MAC) specification. VITA 19.1-1997 Draft D5, June 1997.
- [10] BusNet Link Layer Control(LLC) specification. VITA 19.2-1997 Draft D3, June 1997.