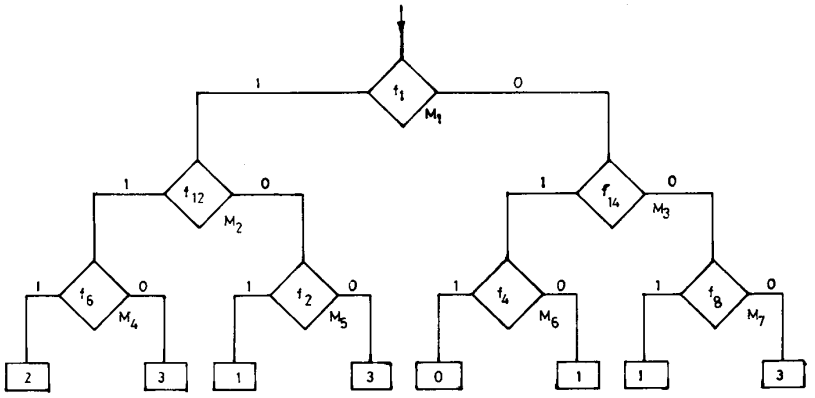

Fig. 6. Reduced BD tree for Example II.

f_1, f_{10}, and f_{11} have the lowest score. Let us choose f_1. At nodes 2 and 3, f_{12} and f_{14}, respectively, have the lowest score. Continuing the process, the reduced BD tree of Fig. 6 is obtained. If only polarity tests are used, four tests will be required on some paths. If f_{10} had been selected at node 1, a reduced tree having some paths with length 4 would have been obtained, because at one of the level 3 nodes an even q_i for one of the output states is split into an odd q_{i+} and q_{i-} by all the npf's. In such cases, it might be worthwhile to go back to the predecessor node and select another test npf.

V. Conclusion

The properties of the class of parity test functions have been considered. The polarity test BD system is a subset of the parity test system. Thus, for a given system, the parity test BD algorithm can generally have less instructions and hence lower average number of cycles to reach an output than the equivalent polarity test algorithm. However, the parity test version requires wider memory, but this may be offset by the fewer number of instructions required. More importantly, the advantage of faster operation while still maintaining programming flexibility will in some cases outweigh the disadvantage of wider words. A heuristic method of reducing parity test based BD trees was also considered. While this method does not guarantee a minimal tree, indications are that it is a useful means of parity tree reduction. Since the method is heuristic, more testing, especially on large systems needs to be done.

BD-based systems are an alternative method of providing solution to logic design problems. They are not meant as a replacement of the other logic methods, but as one more tool for the solution of logic problem. In some cases, the BD solution may be the best one. In the same vein, this paper aims at enhancing the scope of BD systems and the range of tools available for providing BD based solutions to logic system problems.

References

[1] C. Y. Lee, "Representation of switching circuits by binary-decision programs", *Bell Syst. Tech. J.*, vol. 38, pp. 985-999, July 1959.
[2] R. T. Boute, "The binary-decision machine as a programmable controller," *Euromicro Newsletter*, vol. 1, no. 2, pp. 16-22, 1976.
[3] P. J. A. Zsombor-Murray, L. J. Vroom, R. D. Hudson, and T. Le-Ngoc, "Binary-decision-based programmable controllers, Part I," *IEEE Micro*, vol. 3, pp. 67-83, Aug. 1983.
[4] ——, "Binary-decision-based programmable controllers, Part II," *IEEE Micro*, vol. 3, pp. 16-26, Oct. 1983.
[5] ——, "Binary-decision-based programmable controllers, Part III," *IEEE Micro*, vol. 3, pp. 24-39, Dec. 1983.
[6] A. Thayse, "Optimization of binary decision algorithms," MBLE Res. Lab., Brussels, Rep. R348, May 1977.
[7] M. Davio and A. Thayse, "Optimization of multivalued decision algorithms," *Philips J. Res.*, vol. 33, pp. 31-65, 1978.
[8] E. Cerny, D. Mange, and E. Sanchez, "Synthesis of minimal binary decision algorithms," *IEEE Trans. Comput.*, vol. C-28, pp. 472-482, July 1979.
[9] M. Silva and R. David, "Binary-decision graphs for implementation of Boolean functions," *IEEE Proc. E, Comput. Digital Tech.*, vol. 32, pp. 175-185, May 1985.
[10] S. B. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. C-27, pp. 509-516, June 1978.

Transmission Delays in Hardware Clock Synchronization

KANG G. SHIN AND P. RAMANATHAN

***Abstract*—Various methods, both with software and hardware, have been proposed to synchronize a set of physical clocks in the system. Software methods are very flexible and economical but suffer an excessive time overhead, whereas hardware methods require no time overhead but are unable to handle transmission delays in clock signals.**

The effects of nonzero transmission delays in synchronization have been studied extensively in the communication area in the absence of malicious or Byzantine faults. We show that it is easy to incorporate the ideas from the communication area into the existing hardware clock synchronization algorithms in order to take into account the presence of both malicious faults and nonzero transmission delays.

***Index Terms*—Fault-tolerant clock synchronization, malicious faults, phase-locked clocks, transmission delays.**

I. Introduction

The problem of synchronizing a set of physical clocks in the presence of *malicious* or *Byzantine*[1] faults has been studied extensively in recent years [1]-[4]. There are both *software* and

Manuscript received June 22, 1986; revised December 2, 1986. This work was supported in part by NASA under Grants NAG-1-296 and NAG-1-492. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of NASA.

The authors are with the Real Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8718431.

[1] A clock is said to be *malicious if* it *lies* by providing different values to different parts of the system.

hardware solutions to this problem. The software solutions [1], [5] treat the clock values as data values and exchange them periodically in order to ensure proper synchronization. However, as pointed out in [6], this kind of synchronization requires an excessive time overhead, thereby making it impossible to exchange the clock values often enough to achieve a tight synchronization between the clocks.

On the other hand, the hardware solutions [2]–[4] use the principle of phase-locked loops in order to achieve a tight synchronization between the clocks with almost no time overhead. However, unlike the software solutions, none of the existing hardware solutions consider the transmission delays in clock signals. They assume that the transmission delays are small enough to have little effect on the system. This, however, is not valid in a large multiprocessor system, where the physical distance between two clocks could be substantial.

The effects of nonzero transmission delays in phase-locked loops have been studied in great detail in the communication area [7], [8], but not in the presence of malicious faults. In this correspondence, we show that it is easy to incorporate the ideas from the communication area into the existing hardware solutions in order to take into account the presence of *both* malicious faults and nonzero transmission delays.

This correspondence is organized as follows. Section II reviews the basic principle of a phase-locked loop and the problems encountered by it in the presence of nonzero transmission delays. Section III begins with an introduction of the notation to be used and then proposes a hardware solution to overcome the problem of transmission delays. Finally, Section IV presents a brief analysis of the complexity of the solution proposed in Section III.

II. A Brief Review of Phase-Locked Clocks

The basic operating principle of a phase-locked loop is simple. Each clock is an output of a voltage-controlled oscillator. The voltage applied to this oscillator is the output of a phase detector which is proportional to the phase error between the two signals at its input. The two inputs to the phase detector are usually the output of the oscillator it controls and a reference signal with respect to which the oscillator has to be synchronized.

The various hardware synchronization solutions differ from each other in the way the reference signal is chosen. In [2], the reference signal for a four-clock system[2] is chosen to be the median signal among the other three clocks. As evidenced in [3], use of the median signal becomes invalid for a system with more than four clocks. A more complicated means for selecting the reference signal is thus necessary to tolerate more than one malicious fault in the system. In [3], the reference signal is chosen by using a more complicated function on the other clocks in the system. In [4], there is no phase detector. Instead, the control voltage to the oscillator is based on how many of the other clocks are faster than its own output. In the communication area, where malicious faults are not considered, the control voltage fed to the oscillator is directly proportional to the average phase difference between its clock and all other clocks in the system [9].

However, all of the above algorithms are affected by the presence of nonzero transmission delays between the clocks. In the presence of nonzero transmission delays, it is not possible to determine either the relative ordering of the clocks as required in [3] and [4] or the exact phase difference between any two clocks. As a result, none of the above algorithms will achieve the desired objective in the presence of both malicious faults and transmission delays.

III. The Solution

Before presenting our solution to the delay problem in clock synchronization, we need to introduce the following notation.

- N Total number of clocks in the system.
- m Maximum number of malicious faults to be tolerated.
- c_i The signal from clock i.
- T Desired clock period.

[2] To tolerate at most one malicious fault.

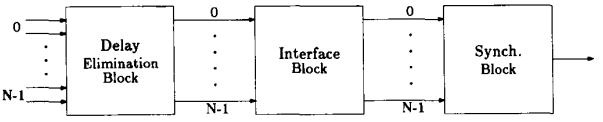

Fig. 1. Block diagram of hardware clock synchronization.

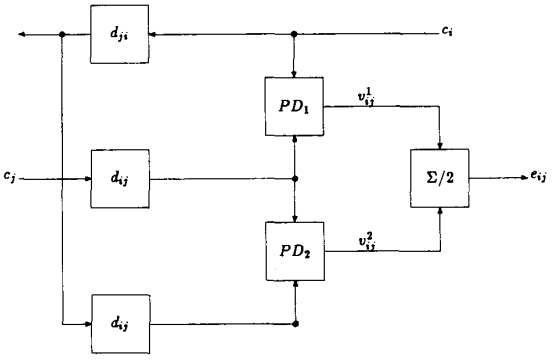

Fig. 2. The jth subblock of the delay elimination block.

- d_{ij} Transmission delay for the signal from clock j to reach clock i.
- τ_{ij} Phase difference between clock i and clock j expressed in time units.
- $\tau_{\max}$ Maximum phase difference between any two nonfaulty clocks in the system, i.e., $\max_{i,j} |\tau_{ij}|$.

The block diagram of the clock synchronization scheme is shown in Fig. 1. It comprises three basic blocks: the *delay elimination* block, the *interface* block, and the *synchronization* block. The delay elimination block receives all the other clocks as inputs and returns analog voltages[3] proportional to the exact phase difference between the input clocks and its own clock. The interface block converts these analog voltages into a form suitable for the synchronization block. The synchronization block could be any one of the existing hardware synchronization circuits mentioned above. The synchronization block will not be discussed here; see [4] and [10] for details of this block.

A. The Delay Elimination Block

For clarity of presentation, consider the delay elimination block in clock i. The delay elimination block consists of $N - 1$ identical subblocks, i.e., one for each other clock in the system. Consider the subblock corresponding to clock j (see Fig. 2). It has two phase detectors and an averager. The inputs to the first phase detector PD_1 are the clock signals c_i and c_j. Since the signal c_j encounters a delay in reaching clock i, the phase difference detected by PD_1 does not represent the true phase difference τ_{ij}.

The inputs to the second phase detector PD_2 are signals c_j and c_i returning from clock j. Due to the transmission delays between the two clocks, the signal c_i returning from clock j will be a delayed version of c_i. Since a delay in time is equivalent to a phase difference, the output of the second phase detector will also depend on the transmission delays. However, if the following four assumptions hold, then it is shown in the theorem below that the average of the outputs of these two phase detectors is proportional to τ_{ij} irrespective of the transmission delays between the two clocks.

- A0. The two phase detectors in each subblock are identical to each other.
- A1. $d_{ij} \approx d_{ji}$ for all i, j.
- A2. $\tau_{\max} < T/2$.
- A3. For all $i, j, nT < d_{ij} < nT + (T/2) - \tau_{\max}$ for some integer n.

[3] one for each of the other clocks.

Assumptions A0 and A1 are easy to implement. A1 can be easily satisfied if the signals c_i, c_j and their returning signals are routed via almost identical paths. A2 is usually a requirement rather than an assumption. Since hardware synchronization algorithms achieve lock step synchronization, $\tau_{max} \ll T/2$. A3 should be treated as a design constraint. Even though two clocks can be physically far apart from each other, they should not be allowed to be at arbitrary distances. The physical distance between any two clocks should be such that the transmission delays satisfy A3.

A3 is not mentioned in [7], because the output of the phase detector is assumed to *always* vary linearly with phase difference. This, however, is not realistic since phase differences between τ and $\tau + nT$, for some integer n, cannot be distinguished from each other by just considering the two signals. Hence, outputs of phase detectors have a sawtooth relationship with phase difference rather than a linear relationship [8]. The need for assumption A3 in the presence of such a sawtooth relationship becomes clear from the theorem below.

Lemma: If integers n_1 and n_2 satisfy the following two conditions

$$\text{a)}\ \frac{-T}{2} \le \tau_{ij} + d_{ji} + n_1 T < \frac{T}{2}$$

$$\text{b)}\ \frac{-T}{2} \le \tau_{ij} - d_{ji} + n_2 T < \frac{T}{2},$$

then $n_1 = -n_2$.

Proof: From A3, there exists an integer n such that $nT < d_{ji} < nT + (T/2) - \tau_{max}$. We now show that $n_1 = -n$ and $n_2 = n$.

By definition, $-\tau_{max} \le \tau_{ij} < \tau_{max}$. Hence,

$$nT - \tau_{max} < \tau_{ij} + d_{ji} < nT + \frac{T}{2} \tag{1}$$

$$-nT - \frac{T}{2} < \tau_{ij} - d_{ji} < -nT + \tau_{max}. \tag{2}$$

Substituting for $\tau_{ij} + d_{ji}$ and $\tau_{ij} - d_{ji}$ from (1) and (2), it is easy to verify that $n_1 = -n$ and $n_2 = n$ satisfy conditions a) and b) (i.e., use A2). Since integers satisfying a) and b) are unique, it follows that $n_1 = -n_2$. ■

Theorem: If the assumptions A0–A3 hold, then the output of the averager $e_{ij} \approx K\tau_{ij}$ for some constant K.

Proof: Let v_{ij}^1 and v_{ij}^2 denote the outputs of the first and the second phase detectors, respectively. Then for a sawtooth phase detector [8],

$$v_{ij}^1 = K_1(\tau_{ij} + d_{ij} + n_1 T) \tag{3}$$

$$v_{ij}^2 = K_2\{\tau_{ij} + d_{ij} - (d_{ij} + d_{ji}) + n_2 T\}$$

$$= K_2(\tau_{ij} - d_{ji} + n_2 T) \tag{4}$$

where K_1 and K_2 are constants and n_1 and n_2 are integers so chosen that $-(T/2) \le \tau_{ij} + d_{ji} + n_1 T < (T/2)$ and $-(T/2) \le \tau_{ij} - d_{ji} + n_2 T < (T/2)$, respectively. From (3) and (4), the output of the averager is

$$e_{ij} = \frac{1}{2}[K_1(\tau_{ij} + d_{ij} + n_1 T) + K_2(\tau_{ij} - d_{ji} + n_2 T)]. \tag{5}$$

From A0, $K_1 = K_2 = K$. From the lemma above, $n_1 = -n_2$. Substituting these relations in (5), we get $e_{ij} = K\tau_{ij} + (K/2)(d_{ij} - d_{ji})$. The theorem then follows from A1.

B. The Interface Block

As stated earlier, the interface block depends on the synchronization block being used. However, for the purpose of illustration, we will consider the interface block for two of the existing hardware synchronization circuits in [1] and [4]. In [4], control voltage to the voltage-controlled oscillator depends on whether more than m clocks are faster than the output of the oscillator, where m is the maximum number of malicious faults to be tolerated. From the description of the delay elimination block, we know that a clock j is faster than a clock i if the voltage e_{ij} is negative. A comparator that outputs a TTL high voltage when the input is negative and a TTL low when the input is positive can be used along with the "greater than m detector" described in [4] to convert the output of the delay elimination block into the desired form.

On the other hand, if a hardware implementation of the algorithm in [1] is being used, then the interface block should chop off all input voltages that are greater in magnitude than $K\tau_{max}$.[4] The outputs of the interface block can then be averaged and used to correct the oscillator just as in [1].

IV. Conclusion

By using the scheme described in this paper, it is possible to incorporate the presence of nonzero transmission delays into the existing hardware synchronization algorithms. However, the proposed scheme increases the complexity of hardware synchronization to some extent. Instead of just $N - 1$ inputs in the existing algorithms, each clock now has $2N - 2$ inputs. This increases the complexity of the interconnection network by almost 100 percent. Also, instead of just one phase detector, the proposed scheme requires $2N - 1$ phase detectors at each clock. This increases the complexity of the synchronization circuitry at each clock.

However, by partitioning the system into clusters, it is possible to reduce the number of inputs to each clock as in [10]. This will not only reduce the total number of interconnections in the system, but also decrease the number of phase detectors required in each clock. As a result, large multiprocessor systems can be synchronized by using the scheme proposed here.

References

[1] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *J. ACM*, vol. 32, pp. 52–78, Jan. 1985.

[2] T. B. Smith and J. H. Lala, "Development and evaluation of a fault-tolerant multiprocessor (FTMP) computer Volume I: FTMP principles of operation," NASA Contractor Rep. 166071, May 1983.

[3] C. M. Krishna, K. G. Shin, and R. W. Butler, "Ensuring fault tolerance of phase-locked clocks," *IEEE Trans. Comput.*, vol. C-34, pp. 752–756, Aug. 1985.

[4] J. L. W. Kessels, "Two designs of a fault-tolerant clocking system," *IEEE Trans. Comput.*, vol. C-33, Oct. 1984.

[5] J. Lundelius and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," in *Proc. Principles Distributed. Comput.*, June 1984, pp. 75–88.

[6] C. M. Krishna, K. G. Shin, and R. W. Butler, "Synchronization and fault-masking in redundant real time systems," in *Dig. Papers, FTCS-14*, 1984, pp. 152–157.

[7] W. C. Lindsey, A. V. Kantak, and A. Dobrogowski, "Network synchronization by means of a returnable timing system," *IEEE Trans. Commun.*, vol. COM-26, pp. 892–896, June 1978.

[8] M. W. Willard, "Analysis of a system of mutually synchronized oscillators," *IEEE Trans. Commun. Technol.*, vol. COM-18, pp. 467–483, Oct. 1970.

[9] H. Inose, T. Saito, and H. Fujisaki, "Theory of mutually synchronized systems," *Electron. Commun. Japan*, vol. 49, pp. 263–272, Apr. 1966.

[10] K. G. Shin and P. Ramanathan, "Clock synchronization of a large multiprocessor system in the presence of malicious faults," in *Proc. 1985 Real-Time Syst. Symp.*, pp. 13–24. (Also in *IEEE Trans. Comput.*, vol. C-36, Jan. 1987.).

[4] Since we know that the maximum skew between any two nonfaulty clocks is less than or equal to τ_{max}.