Maintaining Digital Clocks In Step

Anish ARORA [1] *Shlomi DOLEV* [2] *Mohamed GOUDA* [1]

1. Department of Computer Sciences, The University of Texas at Austin, USA
2. Department of Computer Science, Technion, ISRAEL

Abstract

A system of simultaneously triggered clocks is designed to be stabilizing: if the clock values ever differ, the system is guaranteed to converge to a state where all clock values are identical, and are subsequently maintained to be identical. For an N-clock system, the design uses N registers of $2 \log N$ bits each and guarantees convergence to identical values within N^2 "triggers".

Keywords: stabilization, reliability, distributed algorithms, digital clocks, convergence.

1 Introduction

Digital systems are often designed to be synchronous; that is, a system-wide clock pulse is used to ensure that system parts operate in "lock-step". One building block that is commonly used in the design of such systems is a *digital clock*. Operationally, the task of a digital clock is to maintain a count of the number of clock pulses as they occur.

The use of digital clocks in circuit design is frequent; for example,

- *Synchronization* : A circuit may include some parts that need to synchronize with other parts periodically [1]. The periods, in this case, can be measured using a digital clock.
- *Mode Change* : A circuit may be designed to operate in several modes. For instance, a mode can be allocated to gate asynchronous inputs into registers, and then the register values can be operated upon in a second mode [1]. In such cases, the signal to change the current mode of execution can be generated using a digital clock.

- *Periodic Reset* : A circuit may be designed such that a global signal is periodically generated so as to reset the state of the circuit to some predefined state. In this case, the reset signal can be generated using a digital clock.

In designing circuits where many parts rely on digital clocks, it is preferable to use a number of digital clocks rather than to use only one. This is because using one digital clock may result in long wires which (possibly) traverse multiple chips. It may also result in high fan-out. Consequently, the propagation delay is significant, extra power is needed to drive the signals, and chip space is wasted [2].

The problem of using multiple digital clocks, however, is the added obligation on the circuit designer to ensure that all digital clocks maintain identical values. This obligation is complicated by situations that may set the digital clocks to different values; for instance, powering up the circuit, transient failures, or circuit reconfiguration can all cause different clocks to have different values.

One approach to ensure that all digital clocks maintain identical values is to *compare-and-reset* ; that is, the values of all clocks can be repeatedly compared and, when they differ, all clocks can be reset to some fixed value. One problem of this approach is that it can itself result in long wires and high fan-in and fan-out. Recall that these were the problems that motivated the introduction of multiple clocks in the first place. Another problem is that all clocks are reset even if only one clock differs from the rest. Finally, since the approach is centralized, the overall circuit is no more robust than the reset circuitry itself.

An alternative approach is to design the circuit to be *stabilizing* ; that is, upon starting from an arbitrary state, the circuit is guaranteed within a finite number of pulses to reach a state where all clocks have the same value and are incremented at every subsequent pulse. Based on this approach, we develop in this paper a design for maintaining clocks in step. The design avoids the problems of the previous approach: global comparison and global reset of clock values are not used. Moreover, the design can tolerate the fail-stop failures of individual clocks and interconnecting wires (provided the overall circuit remains connected).

The problem of maintaining clocks in step has been studied earlier by Even and Rajsbaum [3]. They show how clocks can operate in step in a synchronous system where all clocks have identical values initially, but different clocks start counting at different instants. In their design, 'neighboring' clock values may differ by one during early stages of execution, but eventually a state is reached where in step operation resumes. There are, however, two drawbacks in their design. First, if neighboring clock values ever differ by two or more, e.g., due to a transient failure, in step operation cannot be regained. Second, their design employs clocks whose values range over the natural numbers. More recently, Gouda and

Herman have presented a design [4] that overcomes the first drawback, but still employs clocks whose values range over an infinite domain. By way of contrast, our design employs finite-state clocks and so can be used in digital circuits.

The reader should note that the problem of maintaining clocks in step differs from two problems studied earlier, namely, firing squad synchronization [5,6,7] and clock synchronization [8,9]. In firing squad synchronization, every 'participant' in a 'squad' operates in one of two modes; initially, all participants are in one mode, and it is required that if ever some participant initiates a synchronization then subsequently all participants enter the other mode *at the same instant*, simultaneously synchronizing their state. Thus, with clocks as participants, firing squad synchronization suffices to maintain clocks in step as long as different clocks are never in different modes, but does not suffice if different clocks differ arbitrarily.

In clock synchronization, a synchronous system is not assumed: different clocks are triggered by different signals whose rate of drift is assumed to be within known bounds, and it is required to maintain the clocks so that their values are never too far apart. Thus clock synchronization is harder than maintaining clocks in step with respect to the timing of clock updates. On the other hand, existing clock synchronization solutions are not stabilizing: they perform their task under the assumption of some condition holding initially.

2 In-Step Digital Clocks

Consider an undirected and connected graph. Associated with each node u in the graph is

- a *register-set* comprising a finite number of registers; the value of the register-set is defined by a value for each register in the register-set, and ranges over a predefined, finite domain $Q.u$, and
- a *function* whose domain is the cartesian product of $Q.u$ and all $Q.v$ such that the pair (u, v) is an edge in the graph, and whose range is $Q.u$.

A *state* of this system is defined by a value for each register-set.

An *execution* of this system is an infinite sequence of transitions; in each transition, all functions associated with nodes in the graph are simultaneously executed starting from the current state, thereby yielding the next state.

The problem of maintaining digital clocks in step can be stated as follows:

It is required to design a register-set and a function for each node u such that the register-set associated with u includes a register $x.u$ whose value ranges over the interval $0 \,..\, m-1$, where m is a natural number greater than 1, and the resulting system satisfies the following two properties.

1. *Unison* : Starting from any *consensus state*, i.e., one where all x registers have equal values, the value of each x register is incremented by one modulo m in every following transition of the system. (This implies that all states following a consensus state are also consensus states.)
2. *Stabilization* : Starting from an arbitrary state, the system is guaranteed to reach a consensus state.

In Section 3, we present a design that solve the problem posed above. The design enjoys low atomicity: in each transition of the resulting system, every x register is updated based on its value and the x value associated with some neighboring node. Hence, the combinational logic needed to implement the function is easily designed in a modular fashion. Moreover, the design is symmetric: associated with all nodes are identical registers and identical functions (modulo the exact set of neighbors accessed). Hence, the design can tolerate fail-stop failures of nodes and edges in the graph provided the remaining graph is connected.

3 A Fall Back Design

We associate with each node u in the graph a new register $r.u$ whose value ranges over the nodes adjacent to u. In each transition, $x.u$ is updated to $min(x.u, x.(r.u)) +_m 1$, where $+_m$ denotes addition modulo m. Also, $r.u$ is updated to $nxt.(r.u)$, where nxt is a function over the nodes adjacent to u such that upon successive application to $r.u$, nxt returns nodes in some fixed cyclic order. Thus, associated with each node u we have :

register-set of node u ::
  $x.u :\ 0..\, m-1$
  $r.u :\ \{ v \mid (u,v) \text{ is an edge in the graph } \}$

function of node u ::
  $x.u \ , r.u \ := \ min(x.u, x.(r.u)) +_m 1 \ , nxt.(r.u)$

Operationally speaking, the design ensures that each node u scans the clocks adjacent to it in a fixed cyclic order and, in each transition, $x.u$ "falls back" with the clock $x.(r.u)$ being scanned if $x.u > x.(r.u)$; else, it counts normally.

We prove below that our construction solves the problem of maintaining digital clocks in step provided the following inequality holds:

$$\boxed{m > 2 \times deg \times dia}$$

where dia is the diameter of the graph and deg is the maximal degree of a node in the graph. We also show that the convergence span, i.e, the maximum number of transitions needed to reach in-step operation from an arbitrary state, is $3 \times deg \times dia$, and invite the reader to observe that $2 \times deg \times dia < n^2$, where n is the number of nodes in the graph.

Proof of Correctness

Unison. At a consensus state all x registers are equal and, hence, $min(x.u, x.(r.u)) +_m 1 = x.u +_m 1$ for all nodes u and $r.u$. Executing the function of each node in parallel increments each x register by 1 modulo m, thereby yielding the next consensus state.

Stabilization. We prove the stabilization property by considering two cases for the first $deg \times dia$ states in the system execution following an arbitrary state q:

(i) At each of these states, no x register has the value 0.

(ii) At some of these states, some x register has the value 0.

Proof of (i). In the course of this proof, we use the following observation on the persistence of smallest values.

> *Persistence*: Any node whose x value is smallest among all x registers in an arbitrary state also has the smallest x value after one transition is executed, provided that no x register has the value 0 in the resulting state.

Let v be an arbitrary node such that $x.v$ is smallest among all x's at q, and let $dist.u.v$ denote the length of the shortest path from node u to node v in the graph. We claim that upon starting from q and executing k transitions, the resulting state q' satisfies $(\forall u : dist.u.v \leq \lfloor k/deg \rfloor \Rightarrow x.u$ is smallest among all x's). From the claim, it follows that after $deg \times dia$ transitions, the resulting state satisfies ($\forall u : x.u$ is smallest among all x's); i.e, the system is in a consensus state.

We verify the claim by strong induction on k. The base case $(k = 0)$ is trivially true.

For the induction step $(k > 0)$, we note from the induction hypothesis that $(\forall u : dist.u.v \leq \lfloor (k-1)/deg \rfloor \Rightarrow x.u$ is smallest among all x's) holds at the state preceding q'. By persistence, we conclude that $(\forall u : dist.u.v \leq \lfloor (k-1)/deg \rfloor \Rightarrow x.u$ is smallest among all x's) holds at q'. It only remains to show that if $\lfloor k/deg \rfloor \neq \lfloor (k-1)/deg \rfloor$ then $(\forall u : dist.u.v = \lfloor k/deg \rfloor \Rightarrow x.u$ is smallest among all x's) also holds at q'.

Consider any node u such that $dist.u.v = \lfloor k/deg \rfloor$. Due to our choice of the function nxt of u, register $r.u$ has the value w in (at least) one of the preceding deg states, for some node w adjacent to u such that $dist.w.v = \lfloor k{-}1/deg \rfloor$; by the induction hypothesis, $x.w$ has the smallest value among all x's at that state. In the subsequent state, $x.u$ necessarily equals $x.w$; by the induction hypothesis, $x.u$ has the smallest value among all x's ; by persistence, $x.u$ remains smallest in subsequent states up to and including q'.

Proof of (ii). Let v be an arbitrary node such that $x.v$ has value 0 at some state s in the first $deg \times dia$ states following q. We claim that upon starting from s and executing k transitions, the resulting state s' satisfies $(\forall u : dist.u.v \leq \lfloor k/deg \rfloor \Rightarrow x.u \leq k)$. From the claim, it follows that after $deg \times dia$ transitions, the resulting state satisfies $(\forall u : x.u \leq deg \times dia)$; at each of the subsequent $deg \times dia$ states no x register has the value 0, and hence, by the first case, after $deg{\times}dia$ more transitions the system is in a consensus state.

We verify the claim by strong induction on k. The base case $(k{=}0)$ is trivially true.

For the induction step $(k{>}0)$, we note from the induction hypothesis that $(\forall u : dist.u.v \leq \lfloor (k{-}1)/deg \rfloor \Rightarrow x.u \leq k{-}1)$ holds at the state preceding s'. No x register increases by more than one in any transition, hence $(\forall u : dist.u.v \leq \lfloor (k{-}1)/deg \rfloor \Rightarrow x.u \leq k)$ holds at s'. It only remains to show that if $\lfloor k/deg \rfloor \neq \lfloor (k{-}1)/deg \rfloor$ then $(\forall u : dist.u.v = \lfloor k/deg \rfloor : x.u \leq k)$ also holds at s'.

Consider any node u such that $dist.u.v = \lfloor k/deg \rfloor$. Due to our choice of the function nxt of u, register $r.u$ has the value w in (at least) one of the preceding deg states, for some node w adjacent to u such that $dist.w.v = \lfloor (k{-}1)/deg \rfloor$. It follows, from the induction hypothesis and the fact that no x register increases by more than one in any transition, that $x.u$ is at most k in s'.

Convergence Span. We note from the proof of part (i) that, within $deg \times dia$ transitions from an arbitrary state, the system either reaches a consensus state or a state where some $x.u$ is set to 0. In the latter case, we note from the proof of part (ii) that, after $deg \times dia$ subsequent transitions, the system reaches a state where ($\forall$ nodes $u : x.u \leq deg{\times}dia$) holds; a consensus state is guaranteed within the next $deg \times dia$ transitions. In all, therefore, at most $3 \times deg \times dia$ transitions are needed to attain unison. □

4 A Catch Up Design

A strikingly similar design that maintains digital clocks in step is obtained by replacing "*min*" in the function of node u with "*max*". Operationally speaking, this design ensures that in each transition $x.u$ "catches up" with the clock $x.(r.u)$ being currently scanned if $x.u < x.(r.u)$; else, $x.u$ counts normally. Compared to the fall back design, the space and the time complexity of the catch up design is worse: the counter size m must be at least $n \times deg \times dia$, and the convergence span is m. We omit here the proof of correctness of the catch up design, and refer the interested reader to [11].

5 Linear Arrays of Digital Clocks

The fall back design presented in Sections 3 can be easily implemented in hardware. In this section, we describe a hardware implementation of the design for the special case of a linear array of clocks.

Shown below is a schematic diagram of the circuit associated with an intermediate node u. The circuit has a $(\log m)$-bit register that stores $x.u$ (we assume that m is a power of two) and a toggle flip flop $t.u$ that is used as a selector. *Mplus* is a combinational circuit that computes the minimum of two $(\log m)$-bit input operands and increments the result by one modulo m. We assume for correct operation that the delay of the combinational logic exceeds the clock width and is less than the clock period [2].

Each intermediate circuit u operates as follows. At the leading edge of each clock pulse, if $t.u = 1$ holds then $x.u$ is updated to the output produced by the *Mplus* circuit of $x.u$ and $x.(u-1)$; else, $t.u = 0$ holds and $x.u$ is updated to the *Mplus* output of $x.u$ and $x.(u+1)$. Simultaneously, $t.u$ is toggled. In the circuit operation of end nodes, their x register is updated to the *Mplus* of their x value and their (only) neighbor's x value.

As an example, we note that with 8-bit registers clock arrays of up to 64 clocks can be built with the fall back design (since $deg = 2$, $dia = 63$, and $m = 2^8 > 2\times deg\times dia$).

6 Conclusions

We have presented, in this paper, a design for maintaining digital clocks in step. The design is based on the notion of *self-stabilization*. Consequently, the resulting system is self-initializing, does not require global state detection and is robust to transient failures and reconfiguration.

More generally, the concepts discussed in this paper can be extended to reason about the correctness of several classes of reliable systems in a formal manner. In [10], Arora and Gouda present such a formalism based on two notions: closure (which is a safety property analogous to unison) and convergence (which is a progress property analogous to stabilization).

Issues for further research include: What is the least m such that there is a solution for the problem posed in Section 2 in which at most one adjacent clock is "accessed" per clock pulse? Likewise, what is the lower bound on the convergence span for achieving unison? We conjecture that the fall back solution is optimal in these respects.

Acknowledgements. We thank Shlomo Moran, Paul Vitányi and an anonymous referee for helpful comments that improved the presentation of this paper.

References

[1] J.B. Peatman, *Digital Hardware Design,* Addison-Wesley (1980).

[2] C. Mead and L. Conway, *Introduction to VLSI Systems,* Addison-Wesley (1980).

[3] S. Even and S. Rajsbaum, "Lack of a global clock does not slow down the computation in distributed networks", TR522, Computer Science Department, Technion (1988) ; the first part of this paper, "Unison in distributed networks", appeared in *Sequences, Combinatorica, Compression, Security, and Transmission,* E.M. Capocelli (ed.), Springer-Verlag.

[4] M.G. Gouda and T. Herman, "Stabilizing unison", *Information Processing Letters,* Vol. 35, No. 4, pp. 171-175 (1990).

[5] E.F. Moore, "The firing squad synchronization problem", in *Sequential Machines,* E.F. Moore (ed.), Addison-Wesley (1964).

[6] J.E. Burns and N.A. Lynch, "The byzantine firing squad problem", in *Advances in Computing Research: Parallel and Distributed Computing,* Vol. 4, JAI Press, pp. 147-161 (1987).

[7] B.A. Coan, D. Dolev, C. Dwork and L. Stockmeyer, "The distributed firing squad problem", *Siam Journal of Computing,* Vol. 18, No. 5, pp. 990-1012 (1989).

[8] F. Cristian, "Probabilistic clock synchronization", *Distributed Computing,* Vol. 3, pp. 146-158 (1989).

[9] F.B. Schneider, "Understanding protocols for byzantine clock synchronization", Technical Report TR-87-859, Cornell University, Department of Computer Science, Ithaca, NY 14853 (1987).

[10] A. Arora and M.G. Gouda, "Closure and convergence: A formal basis for fault-tolerance", Manuscript (1991).

[11] A. Arora, "A foundation for fault-tolerant computing", Ph. D. Dissertation, in preparation (1991).