Rational Clocking

Luis F. G. Sarmenta*

lfgs@lcs.mit.edu, NE43-609

1 Introduction

An increasingly common problem in designing high-performance computer systems today is that of allowing efficient communication between systems running at different clock speeds. This problem occurs, for example, in uniprocessor systems where the processor runs faster than the bus, and in heterogeneous multiprocessor systems such as the NuMesh [1], where processors of different kinds run at their own maximum speeds. Asynchronous solutions to this problem, which assume the systems' clocks to be completely independent, are typically inefficient because they must allow for sufficiently long synchronization delays to attain acceptable levels of reliability. Synchronous solutions, on the other hand, though more efficient and reliable, have traditionally lacked flexibility because they require the clock frequencies to be related by integer factors.

We present an efficient *and* flexible synchronous solution, called rational clocking, that permits data to be transferred reliably and efficiently between two systems whose clock frequencies are related by the ratio of two small integers. This rational frequency constraint allows a wide variety of frequencies to be used, while assuring a periodic relationship between the two clocks that can be used to determine a schedule for transferring data between the two systems.

2 Rational Clocking

In rational clocking, we use two clocks, Clk_m and Clk_n, with frequencies f_m and f_n respectively, such that

$$\frac{f_m}{f_n} = \frac{M}{N}, \tag{1}$$

where M and N are integers, and such that their active edges are known to coincide at some point in time. The active edges of such clocks will coincide periodically with a period, T_{MN}, equal to

$$T_{MN} = MT_M = NT_N, \tag{2}$$

where T_M and T_N are the two clock periods, and will define *coincidence cycles* equivalent to an integer number of cycles on both systems.

*Supervised by Professors Steve Ward and Gill Pratt, supported in part by ARPA contract number DABT63-93-0008.

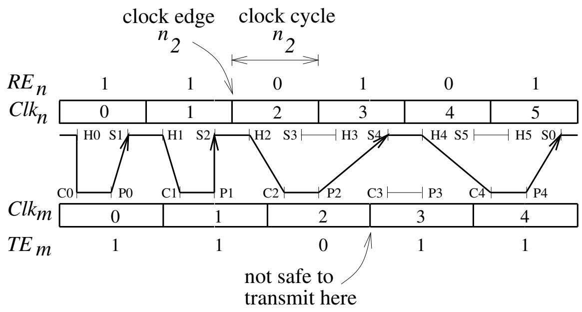

Figure 1: A schedule diagram for $M = 5$ and $N = 6$.

Figure 1 shows a *schedule diagram*, indicating the positions of the two clocks' edges within a coincidence cycle, for the case where $M = 5$ and $N = 6$. With this diagram, we can plot the positions of the transmitter's *transition windows*, i.e., the times during which the transmitter's output may be changing, and the receiver's *decision windows*, i.e., the times during which the register requires data at its input to be stable. This allows us to systematically identify cycles during which it is unsafe to transmit or receive data, and produce a schedule of data transfers that avoids communication errors by inhibiting reception or transmission during the unsafe cycles.

In Figure 1, for example, reception must be inhibited on clock edge n_3 (clock edge 3 of system n) as the new data being transmitted during cycle m_2 is not yet stable by the setup time S_3. Reception can be done on clock edge n_4, but transmission must be inhibited on clock edge m_3 to prevent the transmitter's output from changing while system n is sampling data from cycle m_2. Finally, reception must again be inhibited on clock edge n_5 since new data is not yet available by the setup time S_5.

Figure 2 shows how this idea can be implemented. First, a phase-locked loop (PLL) and two frequency-dividing counters are used to obtain Clk_n from Clk_m and ensure the desired phase and frequency relationships. Then, the values in these counters, which indicate the current clock cycle (within the coincidence cycle) of their respective systems, are used to index lookup tables. Each of these tables produces two control signals per clock cycle number: TE, which controls the transmitting register, and RE, which controls the receiving register. Thus, we can establish

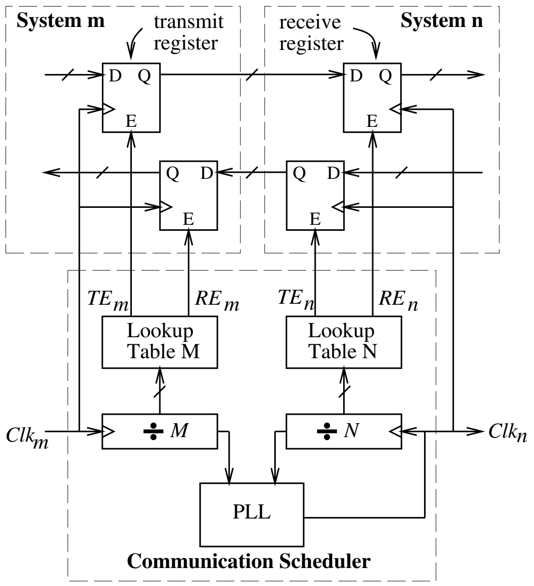

Figure 2: Interface hardware for rational clocking.

reliable and efficient communication between the two systems by simply precomputing the contents of the lookup tables given the desired frequency ratio and the systems' timing parameters. In Figure 1, for example, we set $TE_m[2]$ to 0 to disable transmission on clock edge m_3. We likewise set $RE_n[2]$ and $RE_n[4]$ to 0 to disable reception on clock edges n_3 and n_5 respectively.

3 Double-Buffering

The rational clocking circuit above provides good throughput efficiency[1] for a variety of frequency ratios given typical timing constraints. As the timing windows are made wider, however, throughput not only deteriorates, it also begins to vary widely with the particular frequency ratio being used. This can undesirably limit the usefulness and flexibility of rational clocking in high-performance systems.

This problem occurs because it is possible for one transmission to *contaminate* a previous transmission. In Figure 1, for example, system m cannot transmit on clock edge m_3 because doing so contaminates the data transmitted on clock edge m_2. We can avoid this problem easily by using a *second* transmit register. Whenever transmitting during the current cycle would result in contamination of the transmission from the previous cycle, we place the data in the second transmit register. When the receiver is ready for new data, it then reads from the second register as well. Though simple, this *double-buffering* technique is highly effective and, in almost all practical cases, guarantees 100% throughput efficiency regardless of the specific frequency ratios and timing parameters.

[1] Throughput efficiency is defined as the number of successful data transfers in a coincidence cycle divided by $\min(M, N)$.

4 Run-Time Scheduling

Another problem with the implementation above is that the lookup tables are only good for a specific frequency ratio, and specific timing parameters. We can make a "generic" version of this circuit that works with different ratios and parameters by using a bigger table which takes these other parameters as additional addressing inputs. This table, however, would be impractically large. If we fix the timing parameters and allow the frequency ratio numerator and denominator to vary from 1 to N, the table would need to hold $O(N^3)$ bits. If we further allow the four timing parameters to vary independently, we would need $O(N^7)$ bits.

Fortunately, since the difference between the two clock frequencies is constant, the interacting timing windows will move relative to each other at a constant rate. Thus, it is only necessary to compute their initial positions — from there, subsequent positions can be computed on-the-fly using counters. Using a simple algorithm originally used for line-drawing on raster displays, we can design scheduling hardware that only needs $O(\lg N)$ bits and works for all practical frequency ratios and timing parameters. The remarkable space-efficiency of this hardware makes it possible to design a versatile "off-the-shelf" rational clocking chip, or to integrate the rational clocking hardware into the applications themselves. We can envision, for example, having a microprocessor which can be set to work with synchronous buses of a wide variety of operating frequencies by simply specifying the appropriate values for M and N.

References

[1] S. A. Ward et al. The NuMesh: A Modular, Scalable Communication Substrate. In *ICS '93*.

[2] Computer Architecture Group. "Rationally Clocked Communication." In *MIT Laboratory for Computer Science Progress Report 29*, 1992.

[3] Luis F. G. Sarmenta. *Synchronous Communication Techniques for Rationally Clocked Systems.* Master's Thesis, Department of Electrical Engineering and Computer Science, MIT. May 1995.