Not Asynchronous

Karl Fant
November, 2014

The title is a double negative. Asynchronous is defined to be not synchronous so what can not not synchronous mean? A negative of one possibility covers all other possibilities and if there are more than two possibilities the negative of any one of them, all the non-elephants in the zoo for instance, can be entirely empty of useful content. Is "not QDI[1]" only clocked? Or might it be micropipeline[2], DIMS[3] or NCL[4].[1] We find ourselves lost in an ungrounded and empty referential loop.

Why Synchronous?

Synchronous is defined as occurring at the same time or as having the same period.[5] Our synchronous satisfies both definitions. A clock ticks uniform periods within each of which many behaviors occur at the same time. Is there something fundamental about synchroncity such that all else must be characterized in terms of it, even the non-synchronous? A hint of that something can be found in the limited ability of humans to directly apprehend complexity. To compensate our limited attention span we divide and conquer by breaking complexity down into small parts and understanding how the parts work together. The most straightforward form of this is to characterize the complexity in terms of small steps (a very small number of behaviors at once) and to perform the small steps one at a time sequentially (periodically), as updates to a stable record. One can always understand the progress taken by each small step and one can always see in the record where one is in the journey. So, the gold standard of understanding and confidence, which traces back to the Greeks with their step by step geometric proof, remains one small step at a time: a close relation to synchronicity.

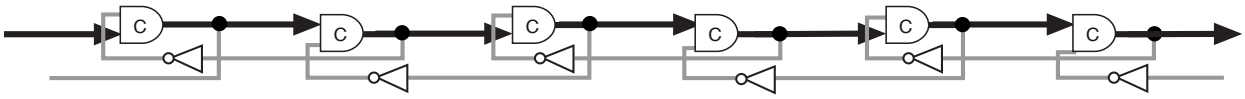
Can we find a positive characterization for our endeavor other than just "not synchronous"? We have to know not what is missing but what are the necessary and sufficient presences. Clockless is often synonymic with asynchronous so we will begin our exploration of "not synchronous", by removing a clock. What have we lost? The clock, a ring oscillator, was the source of liveness. Its rhythmic transitions were the ticks that controlled flows of data through registers.

How do we get our lost liveness and data flow back? The standard answer is that we use handshakes to transfer data between registers? But why do handshakes work and how does handshaking replace a clock? Handshakes are not rhythmic but they do continually transition between request and acknowledge, Each handshake oscillates, and the handshakes link through the registers. And from linked oscillations emerges spontaneous flow.

Linked Ring Oscillators

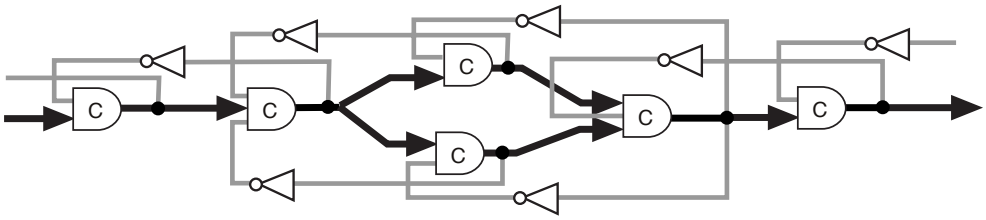
Ring oscillators can be linked with a dual completeness relation such as a C element and with inversion flowing backward in relation to the C elements as shown below. We will call the forward flowing path through the C elements the flow path and backward flowing path containing the inversion the closure path.

1. QDI is Quasi Delay Insensitive, DIMS is Delay Insensitive Minterm Synthesis, NCL is Null Convention Logic.



Alternating wavefronts of values A and B spontaneously and stably propagate through the forward flow path. An oscillation waits on the readiness of a successor oscillation to pass on its A wavefront, after which acceptance, it can accept a waiting B wavefront from its predecessor oscillation, pass it on to its successor and then accept an A wavefront from its predecessor and so on. The oscillators striving to oscillate even when blocked and continuing to oscillate when unblocked provide spontaneously flowing liveness and flow control. Stable wavefronts of alternating A and B values are handed from oscillation to oscillation over the flow path spontaneously pipelining through the linked oscillations. Linked ring oscillators and their pipelining behavior was presented by Ivan Sutherland 25 years ago.[6]

Ring oscillations can be linked in fan-out and fan-in structures forming a network of pipeline flow paths.

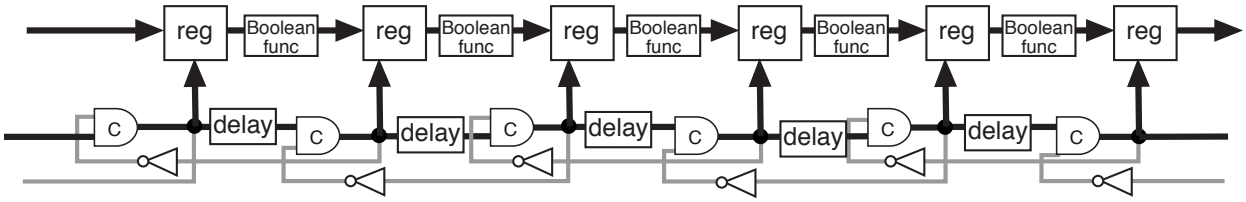


Arbitrarily complex networks of linked ring oscillators can be formed with conditional links. It is a spontaneously flowing network of linked ring oscillators that replaces the single ring oscillator of the clock.

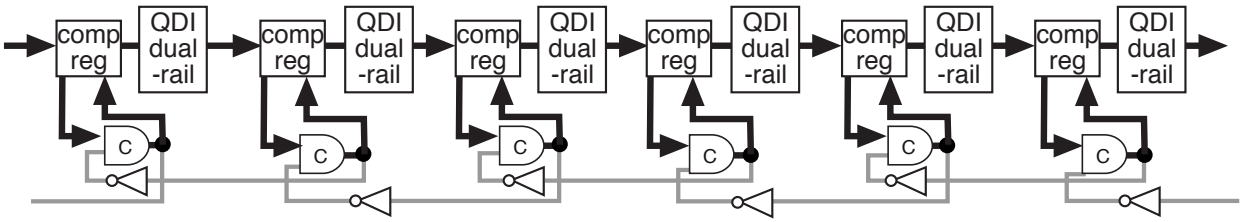
Flowing Computation

The forward flow paths of the network of oscillations can be decorated with computation to realize spontaneously flowing (so called asynchronous) computation.

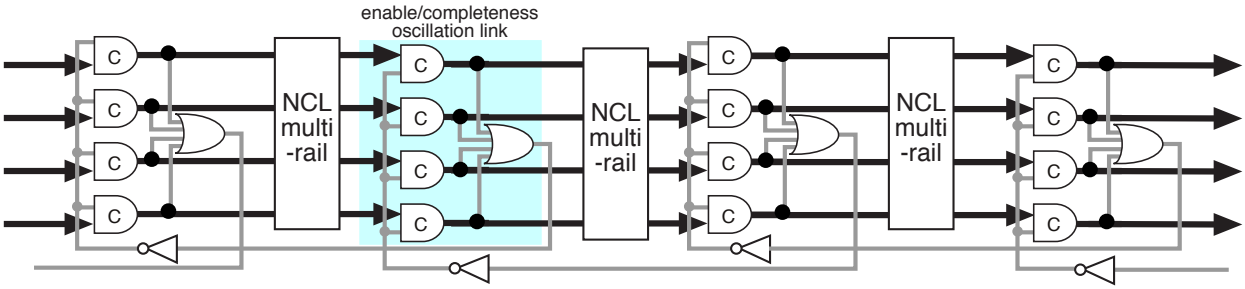
A delay, unique to an associated Boolean combinational computation and identical for both A and B wavefronts, can be inserted into the flow path path of each ring oscillator to determine the completeness of the associated computation. The delayed oscillation signal is then used to "clock" the registers.



One wavefront value A can be assigned to mean data and the other value B can be assigned to mean "not data" (spacer, null) and the combinational computations can be designed with a delay insensitive encoding such as dual-rail employing completeness detecting registers to determine the completeness of computation. In this case the flow path of the oscillation passes through the computation and the registers. The oscillations implement the handshake protocols with the registers.



With the same “data” “not data” wavefront flow computational logic and enable/ completeness logic can be directly integrated in the flow path of the oscillation. The enable/ completeness logic links the oscillations performing the same service as the single C elements in the above. It also maintains flow path values until the values are passed on to the next oscillation performing the same function that a register performs replacing the concept of a controlled register. We have replaced the clock and the registers and even the handshakes that ostensibly replaced the clock with a fully integrated structure of computation and linked ring oscillators specified entirely in terms of logical relationships (no time referent).



The entire flow path can be rendered solely in terms of C elements and OR gates; DIMS essentially. Or the identical functionality can be rendered several times more efficiently in terms of the more general NCL.

The flow network

We have done more than just replace the clock and the registers. With each tic of the clock a state machine guided data through registers weaving a network of computation. Now that weaved network is realized directly as the network of linked oscillations. The sequencing state machine becomes the steering network guiding spontaneously flowing wavefronts of data through the computation.

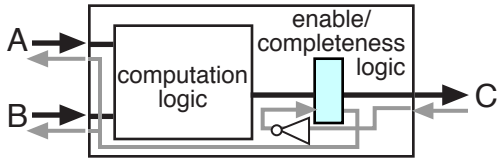
Whither Step by Step Confidence?

The clock was ticking synchronicity step by step but in the network there is no “same period” and there is no “same time”. All the clocked step equivalents take different delays and their delays can overlap in time instead of aligning in time. You never “know” quite where you are in the flowing network. There is no overarching referent, no controller, no stable state to periodically peruse. There is no context other than the local contexts encompassing directly linked neighbors. Have we lost our step by step touchstone of confidence? How can we be sure that our “not synchronous computation” is race free, deadlock free and traffic jam free?

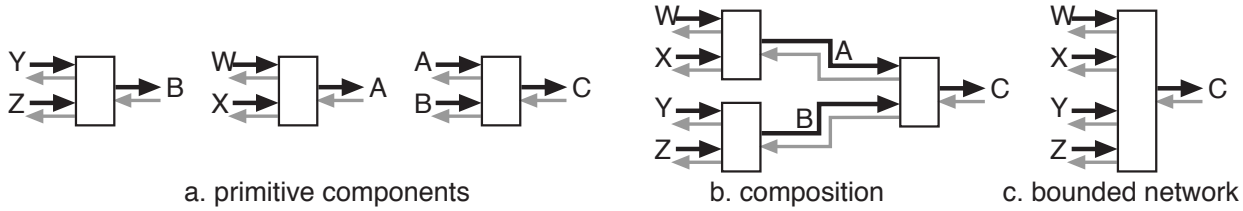
Step by step composition

Instead of operating step by step we satisfy our human sized step by step confidence building by constructing the network step by step insuring at each step that the resulting network behaves

reliably. We begin with primitive network components each of which contains computation logic, enable/completeness linking logic and a boundary of half oscillations with a flow protocol. Each primitive component is simple enough that we can be certain of its behavior in relation to its boundary. For the example below, given data wavefronts presented on A and B a result data wavefront will flow from C. The flow protocol for the A and B inputs is AND related.



We compose primitive components by connecting output half oscillations to input half oscillations. Below in figure a. there are three primitive components similar to the one above. Half oscillations Y and Z are AND related as are W and X as well as A and B The primitive components are composed by connecting A to A and connecting B to B. The result in figure b. is a network with internal oscillations A and B and a boundary of half oscillations W, X, Y, Z and C. The AND protocol of A and B propagates to the new boundary in figure c which is now has protocol (W AND X) AND (Y AND Z). Now given data wavefronts presented on W and X and Y and Z a result data wavefront will flow from C. As long as this boundary protocol is honored the input wavefronts will always flow through the network to C and out.



Synthesis

A network is specified by defining the primitives and then designating their boundary flow relations. Each composition step connects a boundary flow, verifies the compatibility of the connection and tracks the creation and validity of the new boundary. Incompatible boundary connections and problematic boundary evolutions can be detected and reported. The result is a network with a boundary flow protocol that, is confidently race free, deadlock free and live and that flows reliably if the boundary flow protocol is honored. With step by step composition we achieve the same confidence in the flow behavior of the network as we were able to achieve in the step by step behavior of the state machine.

We can build reliably behaving complex flow networks guided by the same specification that would have guided the construction of a clocked state machine or a sequential program. A dependency graph, an algebraic equation, a flow chart or an algorithm might be mapped directly to a flow network with no reference to synchrony (there never is a clock). Or we might use the synchronous resolution steps of an algebraic equation, for instance, to step by step construct a flow network that realizes the equation without synchrony.

The rules of primitive definition, composition and tracking can be codified, the step by step composition and boundary tracking automated and reliable networks of arbitrary complexity can be compiled from specifications of flow relations.

Advantages of network flow computation (the usual)

No clock

Robust

(No critical timing issues, adapts to fabrication variations, correct by construction)

Steering instead of selecting

(do only what is necessary).

Low power

(no clock, no unnecessary computation)

Adapts to actual delays -

(average case behavior, fine grained pipelining)

Insensitive to temperature and voltage variation

(dynamically adapt voltage to performance, subthreshold operation)

Low EMI

Not Asynchronous

Asynchronous is a contentless word and an empty concept that is considerably worse than useless as an intellectual referent.

---

1. A. J. Martin, "Compiling communicating sequential processes into delay insensitive VLSI circuits," Distributed Computing, Vol 1. no 4. pp 226-234, 1986.

2. Ivan E. Sutherland, "Micropipelines", Communications of the ACM, Vol. 32, No. 6, June 1989, pp. 720-738.

3. J. Sparsø, J. Staunstrup, M. Dantzer-Sørenson, Design of delay insensitive circuits using multi-ring structures, in Proc. European Design Automation Conference, 1992, Computer Society Press, Los Alamitos, CA pp. 15-20.

4. Karl M. Fant, Logically Determined Design: Clockless System Design with NULL Convention Logic, (Hoboken, New Jersey, Wiley Interscience, 2005)

5. Webster's New World Dictionary of the American Language, The World Publishing Co., Cleveland and New York, 1957.

6. Sutherland, op. cit., p. 726 fig. 10.