CHAPTER 1

Trusting Logic

Boolean logic is a mathematical symbol system. There is a population of symbols organized into a state space, a set of primitive function mappings, a logic expression specifying a progression of primitive function mappings, and a set of rules of behavior to coordinate the flow of symbols from the state space through the progression of function mappings to resolve the logic expression. This passive symbol system is enlivened by an active agent that can manipulate the symbols in accordance with the rules, traditionally a mathematician with a pencil.

When a logic expression such as Figure 1.1 is enlivened by a mathematician, the logic expression and the mathematician form a complete expression that properly resolves. The mathematician, understanding the rules of behavior, coordinates the behavior of the symbolic expression with her pencil. She animates the symbols moving them from function to function, instantiating functions in the proper order when their input symbols are available. This coordination behavior is embodied in the mathematician's understanding of the rules of interpretation and is not explicit in the logic expression itself.

A function mapping simply specifies a mapping from input symbols to output symbols. Nothing is expressed about the presentation of symbols or about instantiation of the function. A logical expression specifies the dependency relationships among the input and output symbols of functions but expresses nothing about coordination behavior among the functions. On its own expressional merits, without the mathematician, a Boolean logic expression is an incomplete expression that cannot be trusted.

1.1 MATHEMATICIANLESS ENLIVENMENT OF LOGIC EXPRESSION

How the missing expressivity of the absent mathematician is restored to form a sufficiently complete expression is the crux of the matter. One can create a machine to emulate the mathematician. One can supplement the Boolean logic expression with

Logically Determined Design: Clockless System Design with NULL Convention Logic™, by Karl M. Fant
ISBN 0-471-68478-3

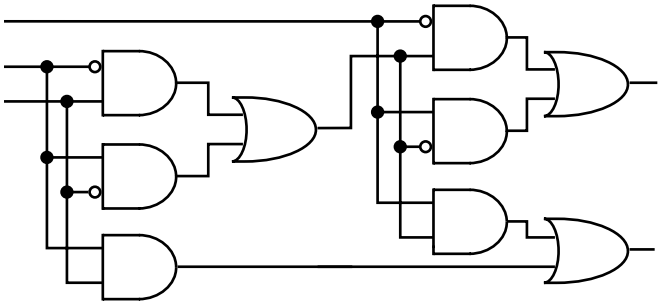

Figure 1.1 A Boolean logic expression.

some other form of expression. Or one can define a logic that is sufficiently expressive in its own terms to fully characterize reliable behavior.

1.2 EMULATING THE MATHEMATICIAN

One might suggest that the mathematician can be replaced by building the rules of coordination into an interpreting machine to properly resolve the logic expression. The logic expression remains a passive symbolic expression that is enlivened by the activity of the interpreter. This is a viable approach for all levels of abstraction except the most primitive. There must be an expression of a most primitive interpreter that cannot itself be interpreted. The logical expression of this last interpreter must behave spontaneously and autonomously on its own merits. The missing expressivity of the absent mathematician cannot ultimately be restored with an interpreting machine.

1.3 SUPPLEMENTING THE EXPRESSIVITY OF BOOLEAN LOGIC

Since the expression of the symbols, the function mappings, and the dependency relationships among functions express no boundaries of instantiation, mapping the logic expression to continuously acting spontaneous behaviors is a valid enlivenment of the symbolic expression. The functions can be represented such that they spontaneously and continuously transform symbols that spontaneously and continuously flow along the dependency relationships among functions. While these continuous behaviors faithfully enliven the logic expression itself, they do not encompass the coordinating expressivity of the missing mathematician.

1.3.1 The Expressional Insufficiency of Boolean Logic

When a new input is presented to a continuously behaving Boolean combinational expression, a stable wavefront of correct result transitions flows from the input through the network of logic functions to the output. Since Boolean functions are continuously responsive to freely flowing symbols and since some functions and signal paths are

faster than others, a chaos of invalid and indeterminate result transitions may rush ahead of the stable wavefront of valid transitions. Such chaotic behavior causes the output of the expression to assert a large number of invalid results before the stable wavefront of correct results reaches the output and the expression as a whole stabilizes to the correct resolution of the presented input. A Boolean logic combinational expression cannot, on its own terms, avoid this indeterminate behavior.

But even if indeterminate transitions could be avoided, there is still no means to determine from the behavior of the expression itself when the output of the expression has stabilized to the correct result. If the current input state happens to be identical to the previous input state, the expression exhibits no transition behavior at all, correct or incorrect. Being unable to express its own boundaries of behavior a Boolean combinational expression cannot coordinate its own successive instantiations nor can it coordinate instantiations with other combinational expressions. These coordination behaviors must be expressed in some other way.

1.3.2 Supplementing the Logical Expression

A continuously behaving Boolean combinational expression can be relied on to eventually settle to a stable correct result of a stably presented input. After presentation of a new instance of input to a Boolean expression, it is necessary and sufficient to wait an appropriate time interval, characterized by the slowest propagation path in the expression, to ensure that the wavefront of stable results has propagated through the expression and the output of the expression has stabilized to the correct resolution of the presented input data. During the time interval all the erroneous transitions due to the racing wavefronts can be ignored, and at the end of the interval the correct result can be sampled. Thus the boundaries of instantiation and resolution can be expressed by supplementing the logic expression with an expression of a time interval. This time interval, however, is a nonlogical expression that must be associated with and coordinated with every Boolean combinational logic expression and that is specific to each implementation of a logic expression.

1.3.3 Coordinating Combinational Expressions

A combination logic expression will receive a succession of inputs to resolve and combinational logic expressions will pass results among themselves to resolve. For a given logic expression, each instance of resolution must be completed and stable before it can be passed on, and the result must be passed on before the given logic expression can accept a new input to resolve. These boundaries of behavior are expressed by the time interval associated with each combinational expression, so the coordination of behavior among combinational expressions must be expressed in terms of these associated time intervals. This coordination is accomplished with a state-holding element between each combinational logic expression that ignores transition behavior during an interval, samples the output of each logic expression at the end of each interval, and stably presents the sampled output as input to a successor combinational expression.

A system consists of multiple combinational expressions, each with an associated time interval and coordinated among themselves in terms of these time intervals. This coordination is most conveniently expressed when all the time intervals are the same duration and are in phase. All logic expressions present valid output and become ready for new input simultaneously, in phase with a common interval expression beating the cadence of successive instantiations.

While the coordinating expressivity of the mathematician has not been restored at the function level, it is sufficiently restored at the level of the time interval to enable practical application.

1.3.4 The Complexity Burden of the Time Interval

Supplementing Boolean logic with an expression of time compounds the complexity of the expression. The structure of the logical expression must be specified. The structure of the time expression must be specified. The delay behavior of the logic in a particular implementation must be determined. The behavior of the time expression in the particular implementation must be determined. The logic expression and the time expression must be integrated. The behavior of the combined expressions must be verified.

There can be no first-order confidence in the behavior of the combined expression. Confidence can only be gained in terms of exhaustive observation of its behavior. At the granularity of the time interval the combined expression generates a sequence of stable states that can be sampled and compared to a theoretically derived enumeration of states to verify correct behavior.

1.3.5 Forms of Supplementation Other Than the Time Interval

There have been ongoing attempts to provide supplemental forms of expression for continuously acting Boolean logic expressions other than the time interval called ‘asynchronous design research.’ The expressional insufficiency of Boolean logic was recognized by Muller in 1962 [35] who introduced two supplementary forms of expression.

The first supplement, called a C-element, could enhance Boolean expressions to avoid indeterminate output. A C-element is a state-holding operator that transitions its output to 1 only when both inputs are 1 and transitions its output to 0 only when both inputs are 0, and for 01 and 10 inputs does not transition its output but holds its current state. The C-element has to be an indivisible primitive operator in its own right. It cannot be implemented in terms of Boolean logic functions. But the C-element, added to Boolean logic, creates a heterogeneous logic that is difficult to formalize, analyze, and synthesize and that still has subtle timing issues [31,1,8].

The second supplement was dual-rail encoding of binary data, in which binary symbols are represented with two wires: 01 represents FALSE and 10 represents TRUE; 00 represents the absence of data and 11 is illegal. Dual-rail encoding can express the absence of data and the presence of data, providing a logical expression of the boundaries of successive data presentations.

The C-element and dual-rail encoding remain standard elements of asynchronous design. The general approach has been to compose Boolean logic expressions enhanced with C-elements, timing assumptions (e.g., wires with zero delay or detailed timing coordination of specific signals within a logic expression) and dual-rail encoding. While this can be achieved to a certain degree, the heterogeneous nature of the expression results in arcane logic structures with subtle modes of behavior including critical timing relationships [4,33,58,59].

Some asynchronous design methodologies pursue the minimal expression of timing relationships. Muller's modules [35], Delay-Insensitive Minterm Synthesis [53], and Martin's methodology [29,30] have approached logically determined design but have not presented a coherent, easily understandable, and adoptable conceptual foundation.

Other methodologies pursue maximum speed and minimum size by minimizing the expression of logical relationships and simply embracing the increased critical timing relationships as an integral part of the design methodology [55,56,54].

The bundled data/micropipeline approach splits the difference. It uses a standard timed Boolean data path and an asynchronous control substrate that expresses the intervals as local clocks to the data path registers using a matched delay element associated with each combinational expression [57,15].

1.3.6 The Complexity Burden of Asynchronous Design

While the C-element and dual-rail encoding were steps in the right direction, they did not step far enough. Boolean logic was retained as a primary form of expression. The result was a heterogeneous expression methodology with subtle logic structures and complex behavior modes that neither enlightened understanding nor enabled practice.

The primary question of asynchronous design—How can Boolean logic be made to work without a clock?—is the wrong question. Boolean logic does not provide an adequate conceptual foundation and simply confuses the issues. The reliance on Boolean logic as a fundamental form of expression is a primary reason that asynchronous design is difficult, is distrusted, and has not been adopted.

1.3.7 The Cost of Supplementation

As logic expression becomes more complex, supplementary expression simply compounds the complexity of expression. Clearly, one should strive to avoid the necessity of supplementary expression.

1.4 DEFINING A SUFFICIENTLY EXPRESSIVE LOGIC

What if a sufficiently expressive logic were available, a logic that could completely express the behavior of a system solely in terms of logical relationships? There would be no need for any supplementary expression with its compounding complexity. Such a logic must symbolically express the boundaries of data presentation, and its operators must appreciate the expressed boundaries.

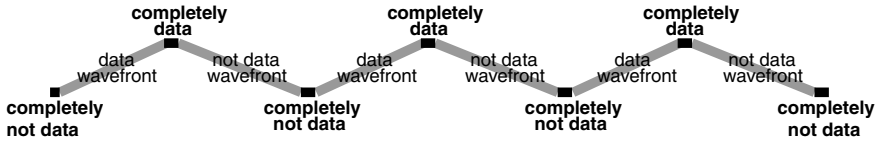

Figure 1.2 Monotonically alternating wavefronts of completely data and completely not data.

1.4.1 Logically Expressing Data Presentation Boundaries

First, the representation of the data must include an explicit symbolic expression of 'not data'. A symbol that explicitly means 'not data' is added to the 'data' symbols, and the input presented to the logic operators transition monotonically between 'completely not data' and 'completely data', as shown in Figure 1.2. The transition of the input from 'completely not data' to 'completely data', called a 'data' wavefront, expresses a new data presentation and the beginning of an instance of resolution. The transition of the input from 'completely data' to 'completely not data', called a 'not data' wavefront, expresses the end of an instance of resolution and the boundary between instantiations. Dual-rail encoding is a specific instance of such a representation.

1.4.2 Logically Recognizing Data Presentation Boundaries

To logically appreciate the presentation boundaries the logic operators must respond only to completeness relationships at their inputs in contrast to continuously responding to all possible inputs.

If input is:

- 'completely data', then transition output to the correct 'data' resolution of input,
- 'completely not data', then transition output to 'not data',
- neither 'completely data' nor 'completely not data' do not transition output.

Example operators are shown in Figure 1.3. T and F are the 'data' symbols and N is the 'not data' symbol. The dash means that the output of the operator does not transition.

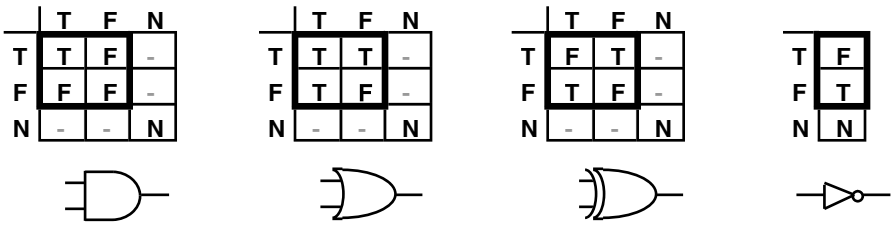

Figure 1.3 Logic operators that recognize 'data' from 'not data'.

Each logic operator, responding only to completeness of input relationships, coordinates its own behavior with the completeness behavior of the presented input. When the output of a logic operator monotonically transitions to 'completely data', it means that the input is 'completely data' and the data output is the correct result of the presented input. When the output of a logic operator monotonically transitions to 'completely not data', it means that the input is 'completely not data' and the 'not data' output is the correct result of the presented input. The C-element is a specific instance of such a logic operator.

This completeness behavior of the logic operator scales up for any acyclic combination of logic operators. Consider the combinational expression shown in Figure 1.4. Circles are logic operators and they are acyclically connected from input to output. Divide the network arbitrarily into N ranks of operators ordered progressively from input to output, with all inputs before the first rank and all outputs after the last rank. The rank boundaries are shown in Figure 1.4 with vertical lines labeled alphabetically in rank order from input to output. Consider that the expression is in a completely 'not data' state and a 'data' wavefront is presented. The 'data' wavefront will transition the inputs to 'data' as the input monotonically transitions to 'all data'.

- For the symbols crossing E to be all data all of the symbols crossing D must be data.
- For the symbols crossing D to be all data all of the symbols crossing C must be data.
- For the symbols crossing C to be all data all of the symbols crossing B must be data.
- For the symbols crossing B to be all data all of the symbols crossing A must be data.
- Therefore, for all the symbols after E to be data, all the symbols before A must be data.

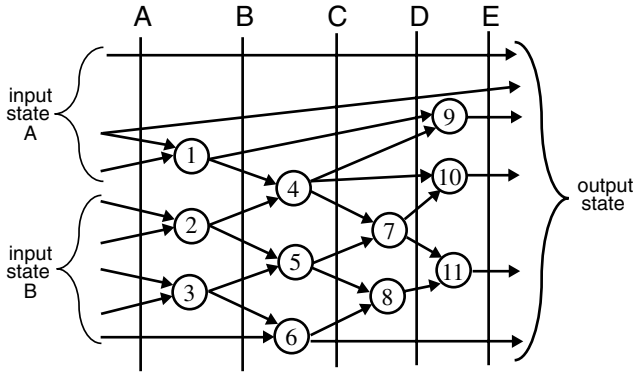

Figure 1.4 The completeness criterion for a combinational expression as a whole.

These considerations are also true for the ‘not data’ wavefront presented when the expression is in a ‘completely data’ state. Simply substitute ‘not data’ for ‘data’ in the text above.

The output of the combinational expression as a whole maintains the monotonic behavior of the input. When the output transitions to ‘completely data’, it means the input has transitioned to ‘completely data’ and the output is the correct resolution of the input. When the output transitions to ‘completely not data’, it means the input has transitioned to ‘completely not data’ and the output is the correct resolution of the input.

Given the monotonic behavior of the input, it does not matter when or in what order transitions occur at the input of the expression. Nor does it matter what the delays might be internal to the expression. Consider operator 7 in Figure 1.4. It does not matter how long the ‘data’ symbols (‘not data’ symbols) take to propagate through other operators and over signal paths to the input of operator 7, its output will not transition until all symbols are ‘data’ (‘not data’) at the input of the operator. For each wavefront, each logic operator synchronizes its input and switches its output exactly once coordinating the orderly propagations of monotonic transitions to correct result symbols through the combinational expression until the output of the expression as a whole is complete. There are no races, no hazards, and no spurious result symbols during the propagation of the monotonic wavefront of correct results through the combinational expression.

The behavior of the combinational expression is expressed entirely in terms of logical relationships. No expression of any time relationship or any other nonlogical relationship is necessary to fully characterize the behavior of the expression. The behavior of the combinational expression is completely logically determined.

1.4.3 Logically Coordinating the Flow of Data

Now that the boundaries of presentation and resolution can be expressed logically in terms of completeness relationships, the flow of presentation wavefronts among expressions can be coordinated in terms of these same completeness relationships.

When a combinational expression detects completeness on its output it generates an acknowledge signal in the inverse domain from the detected completion. For ‘completely data’ output the acknowledge signal transitions to ‘not data’. For ‘completely not data’ output the acknowledge signal transitions to a ‘data’ symbol.

Each combinational expression includes an acknowledge signal as an integral part of its input completeness relation. When the acknowledge signal presented to a combinational expression transitions from ‘not data’ to ‘data’, the input of the combinational expression will become completely data and a data wavefront will be enabled to propagate to the output of the combinational expression. As long as the input acknowledge signal remains ‘data’ the combinational expression will stably maintain its ‘data’ output symbols even if a ‘not data’ wavefront is presented on the data path. When the acknowledge signal transitions from ‘data’ to ‘not data’, a ‘not data’ wavefront will be enabled to flow through the combinational expression. As long as the input acknowledge signal remains ‘not data’, a combinational

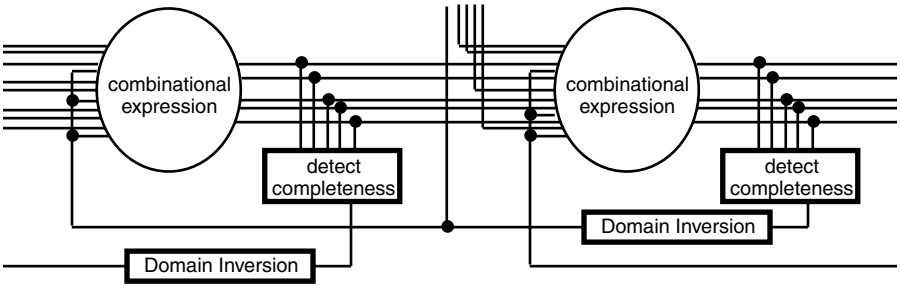

Figure 1.5 Coordination among combinational expression in terms of completeness relationships.

expression will stably maintain its 'not data' output symbols even if a 'data' wavefront is presented on the data path.

When a combinational expression detects completeness on its output, it means that a wavefront has been enabled through it by the acknowledge signal and that it is stably maintaining the output wavefront. Upon output completeness it can inform, via the acknowledge signal, each combinational expression whose output provided input to it that they need no longer maintain their output and that they can allow a new input wavefront to propagate. This relationship is illustrated in Figure 1.5. Stably maintained wavefronts flow through the system passed from combinational expression to combinational expression fully coordinated in terms of completeness relationships.

1.4.4 Mathematicianless Completeness of Expression

Symbols move from function to function, instantiating in the proper order as their input symbols are available. Wavefronts flow from expression to expression and instantiate in the proper order when their inputs are complete. This coordination behavior is explicit in the logic itself. The expressivity of the absent mathematician has been fully restored in the expressivity of the logic.

A logic expression can be complete and sufficient in itself. No supplementary expression is required.

1.5 THE LOGICALLY DETERMINED SYSTEM

A logically determined system is a structure of logical relationships whose behavior is completely and unambiguously determined by those logical relationships. Wavefronts spontaneously flow through combinational expressions fully coordinated by logical relationships. One might try to imagine the flowing wavefronts in terms of a single collective state, but this imagined collective state is subject to concurrent transitions coordinated solely by logical relationships with no consideration for any time relationships. The behavior of the collective state in relation to time is

simply indeterminate. There is no way of sampling a stable configuration of the collective state that at any chosen instant might be in transition. The familiar methodology of understanding, and acquiring confidence by enumerating and comparing sequences of a collective state, is not applicable to a logically determined system.

1.6 TRUSTING THE LOGIC: A METHODOLOGY OF LOGICAL CONFIDENCE

Behavior that appears chaotic, undetermined, unknowable, and untrustable in the context of a collective state appears orderly, completely determined, directly knowable, and trustable in the context of logic relationships. The only path to understanding and trusting the behavior of a logically determined system is in terms of the logic relationships.

The behavior of a first logical expression can be understood locally in terms of its direct neighbor expressions. As the neighbor expressions are understood, their behavior neighborhood includes the first logical expression. As understanding progresses logical expression by logical expression, a tapestry of logically determined behavior is woven of overlapping behavior neighborhoods, Confidence in system behavior is progressively approached in humanly graspable steps.

1.7 SUMMARY

Boolean logic is insufficiently expressive on its own terms and must be supplemented with an expression of a mathematician or an expression of time or an expression of non-Boolean operators. The supplemental expression compounds the inherent complexity of the logical expression. The behavior of a supplemented expression cannot be directly trusted, so confidence in system behavior is found in exhaustive correspondence with enumerated states.

While the C-element and dual-rail encoding were steps toward sufficient expressivity, they did not step far enough. The C-element prefigures the importance of completeness behavior and dual-rail encoding prefigures the more general concept of monotonic transitioning between ‘data’ and ‘not data’. The halter limiting the steps was the continuing reliance on Boolean logic as a primary form of expression.

Stepping far enough leads to a coherent logic that is sufficiently expressive to completely and unambiguously express the behavior of a system solely in terms of logical relationships. There is no need for any supplemental form of expression.

A system operating solely in terms of logical relationships behaves generally concurrently. There is no temporal synchronization of state transitions and, hence, no collective state that is reliably and meaningfully samplable. What appears to be chaotic and disorderly from the point of view of the state behavior is fully determined and orderly from the point of view of the logic. Confidence in system behavior requires a new rationale grounded in trusting the logic. A quite different regime of design, behavior, and rationale of confidence emerges. Since the behavior

of a design is completely determined by the logical relationships, the logic can and must be trusted.

1.8 EXERCISES

1.1. Discuss the philosophical implications of a concept which is considered a primitive characterization but which must be supplemented to fulfill its mission of primitivity.