Theseus Logic

Setting the Standard for Clockless Systems™

NULL Convention Logic™

Karl M. Fant, Scott A. Brandt
Theseus Logic, Inc.
1080 Montreal Ave., #200
St. Paul, MN 55116-2325
(651) 699-6622

NULL Convention Logic™

Karl M. Fant
Theseus Logic, Inc.
1080 Montreal Ave, Suite 200
St. Paul, Minnesota, USA 55116

Scott A. Brandt
Dept. of Computer Science
Campus box 430
University of Colorado
Boulder, Colorado, USA 80309

Abstract

NULL Convention Logic™ (NCL™) is a symbolically complete logic which expresses process completely in terms of the logic itself and inherently and conveniently expresses asynchronous digital circuits. The traditional form of Boolean logic is not symbolically complete in the sense that it requires the participation of a fundamentally different form of expression, time in the form of the clock, which has to be very carefully coordinated with the logic part of the expression to completely and effectively express a process. We introduce NULL Convention Logic™ in relation to Boolean logic as a four value logic, and as a three value logic and finally as two value logic quite different from traditional Boolean logic. We then show how systems can be constructed entirely in terms of NULL Convention Logic™.

1. Introduction

NULL convention Logic™ [7] is derived directly from the Invocation Model of Process Expression. The Invocation Model is a conceptual model of general process expression in contrast to a model of computation. It transcends limiting mathematical notions of computation to provide a unifying conceptual framework which relates all forms of process expression from the simplest physical and chemical processes to the most complex natural and artificial processes. For instance, the processes of cell metabolism and of digital computers are characterized in terms of the same concepts and relationships. They simply occupy different places in a single expression space defined by the Invocation Model.

A central concept of the Invocation Model is the **general notion of completeness**. Just how fundamental the notion of completeness is will become clear in the course of the discussion. It will appear in many guises in different aspects of process expression but always in service of providing necessary and sufficient conditions. We will begin with the notion of **symbolic completeness of expression**. symbolically complete expression is complete in and of itself solely in terms of the relationships among symbol values in the expression. It integrates the expression of data transformation and what is generally viewed as the expression of control in a single symbolically

determined expression, without appeal to any extra-symbolic expression such as a clock or a controller. Processes in nature are symbolically complete, resolving spontaneously and autonomously without appeal to any global time authorities or control sequencers. The artificial processes devised by humans can also be symbolically complete.

Traditional Boolean logic is not symbolically complete. Among its component elements, a traditional Boolean logic circuit exhibits time dependent relationships as well as symbolic-value-dependent relationships. The symbolic-value-dependent relationships depend on the interconnection of the logic gates and their truth tables. The time-dependent relationships depend on the propagation delay times of the component elements to express the validity of data values and the invalidity of data values. These two aspects of expression are independent of each other in that the time relationships can be expressed quite arbitrarily in relation to the expression of symbolic relationships and vice versa. These two quite independent and partial expressions must be carefully and explicitly coordinated to provide a complete correctly resolvable expression of a process. A carefully engineered Boolean logic circuit with its clock is a complete expression and can be made to work, but it is not a symbolically complete expression.

Traditionally, in computer science, the expression of data transformation and the expression of control have been viewed as inherently independent aspects of process expression which must, of necessity, be carefully coordinated. In programming languages this manifests as explicit sequence control of assignment statements. In traditional Boolean logic circuits, the gates and their interconnections are the data transformation aspect and the timing relationships expressed by careful engineering and the clock are the control aspect (which expresses the validity and invalidity of data values). But, the data transformation aspect and the control aspect of process expression are not inherently independent. The expression of both aspects can integrated into a single expression purely in terms of symbolic-value-dependent relationships with no external control expression at all. This is what is meant by a **symbolically complete expression**. It is completely expressed and completely determined solely in terms of symbolic-value-dependent relationships. A symbolically complete logic circuit would have no time relationships at all and would be completely **insensitive to the propagation delays among its component elements**.

There have been attempts to eliminate time dependencies in digital logic circuits since D. E. Muller, pioneered the pursuit in the late 1950s [1,2]. These attempts (in order of increasing independence from time issues) are referred to as fundamental mode circuits, speed-independent circuits, and delay insensitive circuits. Only the delay insensitive circuits are completely free of delay issues of all circuit components including gates and wires. Delay insensitive circuits are generally considered the most difficult, expensive and elusive circuits to design. Only a few truly delay insensitive circuit designs are known. Fundamental mode circuits typically use matched delay lines to provide a local time reference for each circuit [4] and speed independent circuits must make assumptions about the insignificant propagation delay of the wires in the circuit.

These attempts to eliminate time dependencies are nearly always expressed within the traditional context of Boolean logic. They focus on designing Boolean logic circuits with appropriate switching behavior and surrounding them with Muller C-elements to express the control, then transmitting data between circuits with dual rail encoding. These structures of Boolean logic circuits and C-elements can become very subtle, very large and very expensive[3].

There has been much recent work on asynchronous design [5,6] but, while the pursuit of asynchronous circuits has produced many interesting results, it has not delivered a theoretically

complete and economically feasible solution. The current approaches still require some assumptions about local transmission delay and the extra circuitry needed to achieve asynchronous control is considerably more than is required by a functionally equivalent clocked Boolean logic circuit.

NULL Convention Logic™ is a theoretically complete and economically feasible approach to delay insensitive circuits. In this paper we first introduce the NULL Convention in the context of Boolean logic, showing how to make Boolean logic symbolically complete as a four value logic. Then, we show how the NULL Convention can be implemented as a two value logic, which will prove to be the most practical form. We conclude with a discussion of the properties of NULL Convention Logic™.

2. Making Boolean Logic Symbolically Complete

The invocation model teaches that to achieve symbolic completeness of expression, both data transformation and control must be expressed in the most primitive inherently enforced **mutually exclusive value assertion domain** of the expression. In a mutually exclusive value assertion domain, only one of a set of two or more values can be asserted at a time. Every expression environment has a primitive mutually exclusive value assertion domain. For Boolean logic the primitive mutually exclusive value assertion domain consists of True and False, only one of which can be asserted at a time. For the decimal number system the values 0 through 9 are mutually exclusive. In both of these cases the values comprising the mutually exclusive assertion domain express only data meanings. **There is no value in either domain that expresses a control or non-data meaning**. The Invocation Model teaches that to make Boolean logic symbolically complete, we must add a value to the primitive mutually exclusive assertion domain.

Making Boolean logic symbolically complete will be presented in two steps. The **first step** is to assign a value in the most primitive mutually exclusive assertion domain to express the validity / invalidity of data. To retain the sense of Boolean logic, we will retain the two values expressing data meanings in the primitive domain, so the first step is to add a third value to the mutually exclusive value assertion domain which we will call **NULL** (not a valid data value). We then define the truth tables to assert a data value (True, False) only when both input values are data values and to assert a NULL value otherwise. Figure 1 shows the NULL value added to the traditional Boolean truth tables and specifies the behavior of these preliminary NULL Convention Logic™ gates.

	T	F	N
T	T	F	N
F	F	F	N
N	N	N	N

AND

	T	F	N
T	T	T	N
F	T	F	N
N	N	N	N

OR

T	F
F	T
N	N

NOT

Figure 1. Boolean truth tables with NULL value added.

We now have a **three value logic**. Each wire can assert one of three values (T, F or N) and each gate resolves three input values. The values T and F will be collectively referred to as data values and

NULL will be referred to as a non-data value. If a data value is asserted on a wire, the data value is valid. A transition from NULL to a data value marks the beginning of validity. A transition from a data value to NULL marks the end of validity or the beginning of invalidity.

These truth tables enforce the **completeness of input criteria** for data. This means that a gate will assert a data result value only when a complete set of data values is present at its input. In this case, a complete set of input data values is for both inputs to be data (either TRUE or FALSE, but not NULL). Furthermore the asserted result data value is the correct resolution of the presented input data values If either or both input values are NULL a NULL result value is asserted.

THE DATA RESOLUTION WAVE FRONT

The completeness of input criteria is the key to speed independent logic circuits. The completeness of input criteria for each gate scales up for combinational circuits as a whole. Consider Figure 2.

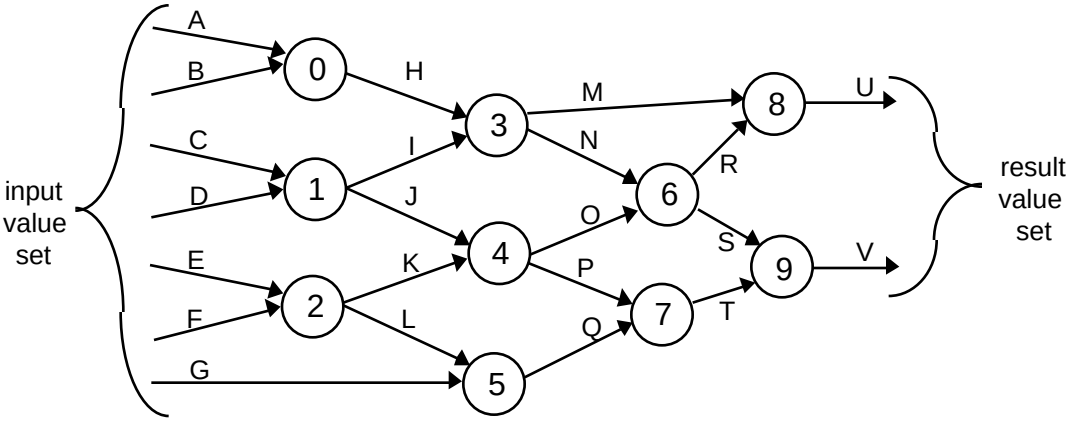

Figure 2. Example combinational circuit.

Assume that the circuit is in an all NULL state in that all of the input values, internal values and result values are all NULL. If one input value, A for instance changes to data, gate 0 will continue asserting a NULL value. Gate 0 does not assert a data value until both input values for A and B are data. To see how this behavior scales up for the whole circuit consider gate 6. Gate 6 will not assert a data value until both inputs N and O are data at the gate . It doesn't matter how long it took for the data values on N and O to propagate to the gate 6 input or in which order they arrive. When both input values are data at gate 6, it asserts the correct result data value. This transition from NULL to data also asserts the validity of that result data value and asserts the completion of gate's resolution entirely symbolically. No other aspect of expression and no time relationship is associated with the assertion of these three facts. Each gate is a **synchronization node** managing an orderly wave front of correct result values propagating through the circuit.

The circuit as a whole will not assert a complete set of result data values until there is a complete set of input data values and the resolution of the input data values has propagated through the circuit. If one input value remains NULL at least one result value will remain NULL. For instance, if all the input values are data except G, which remains NULL, then gate 5 will assert a NULL result value for Q, gate 7 will assert a NULL result value for T and gate 9 will assert a NULL

result value for V. When G becomes data, the input data set is complete. The data value will propagate through the circuit and V will become data and the result data set will be complete (all datas, no nulls).

When all the result values become data, it means that a complete set of input data values are presented to the circuit, that the resolution of the presented input value set is complete and the asserted set of result values is the correct and valid resolution of the presented set of input data values. This is what is meant by the completeness of input behavior of each gate scaling up for a circuit as a whole. The circuit does not assert a complete set of result data values until a complete set of input data values are present at the input to the circuit and resolution of the input values is complete. **The circuit as a whole enforces the completeness of input criteria for data.**

The completion of resolution of an input data set can be determined by simply monitoring the result values. When the result values transition from all NULL to a complete result data value set (all data at the output), then the resolution of a complete set of input data values is finished. **The circuit indicates its own completion of resolution, autonomously and purely symbolically**. No external expression or authority such as a clock, delay line or controller is needed.

THE NULL -DATA CYCLE

To express the completeness of input criteria for data, the circuits have to start out in an all-NULL state. After each data set resolution the circuit must be returned to an all NULL state before the next data set is presented for resolution, so the circuit must cycle alternately between the all NULL state and the data resolution state. Now we must determine when the circuit enters the all NULL state from a data resolution state. If we can determine when the circuit is in an all NULL state, then we will know when it is ready for a new set of input data to be resolved.

It can be seen from the truth tables of Figure 1 that the completeness of input criteria is <u>not</u> enforced for NULL values in relation to data values. If one input value is NULL then the result value will become NULL. Referring to Figure 2, it is possible for a single input value to become NULL and drive all the result values to NULL. For instance if D becomes NULL the NULL value will propagate through gates 1, 3, 4, 6, 7, 8 and 9 driving all of the result values to NULL while there are still data values lingering on the input and internal to the circuit.

THE NULL WAVE FRONT

The **second step** in making Boolean logic symbolically complete is for the gates to enforce the completeness of input criteria for NULL in relation to data (as well as for data in relation to NULL). This can be accomplished in two ways.

1. Another (i.e. 4th) value can be added to the primitive mutually exclusive value assertion domain of the logic.
2. A feedback variable can be added to each gate

THE INTERMEDIATE VALUE SOLUTION

Adding another value to the primitive mutually exclusive value assertion domain of the logic provides a solution that is purely delay insensitive and purely symbolically complete. We will add another value called the Intermediate value and configure the truth tables as shown in Figure 3.

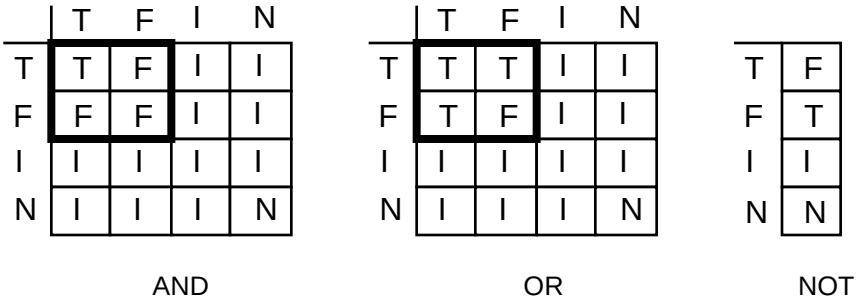

Figure 3. Intermediate value truth tables.

We now have a **four value logic**. It can be seen from the truth tables that a gate will only assert a data result value when both input values are data, and will only assert a NULL result value when both input values are NULL. The truth tables directly enforce the completeness of input criteria for both data and NULL. **When the result values are all NULL the inputs are all NULL and circuit is completely reset and ready for a new data set. When the result values are all data, a complete data set is presented at the input to the circuit and its resolution is completed.** The result values transition from all NULL through Intermediate values to all data then from all data through Intermediate values to all NULL. The watcher of the result values looks for all data and all NULL at the output and simply ignores Intermediate values.

Again, the behavior of each gate scales up for the circuit as a whole. Refer to Figure 2 again and assume each gate is an Intermediate value gate. Beginning with the circuit in an all data state, if only D becomes NULL gate 1 will assert an intermediate value until C also becomes NULL. If a single input value remains data there will be at least one result value that remains Intermediate. When all result values become NULL it means that the input values are all NULL and the NULL values have propagated through the circuit and that the circuit is in a completely NULL state. **The circuit as a whole enforces the completeness of input criteria for both data in relation to NULL and for NULL in relation to data**.

It can now be determined when the circuit is completely reset to NULL and ready to accept a new input data set to resolve by simply monitoring the result values. When the result values transition from a complete result data set to all NULL the circuit is completely reset and ready to accept a new data set. **The circuit indicates its own readiness to accept a new input data set purely symbolically and autonomously**. No expression or authority external to the circuit expression such as a clock, delay line or controller is needed.

We now have a circuit that is a complete expression in and of itself. It can tell the world when it is ready to accept data to resolve and it can tell the world when it has completed a data resolution. Data resolution occurs in an orderly wave front of correct result values within the circuit. It can be a fully autonomous and asynchronous element of a larger whole (i.e. a system).

An intermediate value logic circuit is a symbolically complete process expression and is purely symbolically determined. The Intermediate value solution is a theoretically complete and general solution to delay insensitive circuit synthesis. Its symbol resolution behavior is not affected in any way by the propagation delay of any element in an expression.

The addition of the NULL value or the Intermediate value did not change the transform specifications for the data values. Intermediate value NULL Convention Logic™ gates can replace the gates of a standard Boolean logic combinational circuit one for one, and the circuit will provide the identical logic function as before. It will simply resolve in a more orderly manner and assert its own completion, as well as its own readiness to accept a new input data set.

THE FEEDBACK SOLUTION

The feedback solution makes each gate a state machine with hysteresis of result value assertion. Figure 4 shows the truth table for the feedback gate. Do not be daunted by the seeming complexity of this table. This is just the first example to introduce the NULL convention.

	R	N									F									T							
INPUT	A	N			T			F			N			T			F			N			T			F	
	B	N	T	F	N	T	F	N	T	F	N	T	F	N	T	F	N	T	F	N	T	F	N	T	F	N	T	F
RESULT	R	N	N	N	N	T	F	N	F	F	N	F	F	F	F	F	F	F	F	N	T	T	T	T	T	T	T	T

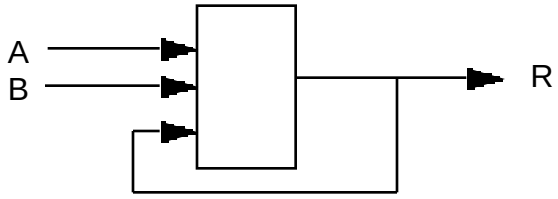

Fig 4. The feedback gate and its truth table.

R is the result variable fed back to the input. When the gate is asserting NULL it is in the NULL (N) state and will continue asserting N until both input values become data (T,F) at which point it will transition its result value to a data value and enter a data state. When the gate is in a data state (R = F or T) it will continue asserting a data value until both input values become NULL (N) at which point it will transition its result value to N and enters a null state. **The feedback gate enforces the completeness of input criteria for both data in relation to NULL and for NULL in relation to data.**

Again, the behavior of each gate scales up for the circuit as a whole. Refer to Figure 2 again and assume each gate has feedback. Beginning with the circuit in an all data state, if only D becomes NULL gate 1 continues to assert a data value until C also becomes NULL. If a single input value remains data there will be at least one result value that remains data. When all result values become NULL it means that the input values are all NULL and the NULL values have propagated through the circuit and that the circuit is in a completely NULL state. **The circuit as a whole enforces the completeness of input criteria for both data in relation to NULL and for NULL in relation to data**.

A NON-CRITICAL TIME RELATIONSHIP

While the intermediate value solution uses play through gates with no time relationships at all and is fully delay insensitive, the feedback solution is not fully delay insensitive in that there is a non-critical time relationship involved. The feedback path around each gate must stabilize faster than successive wave fronts of transition pass through the circuit as a whole. We call this a non critical time relationship because it is easy to achieve since the circuit propagation time will typically be much longer than the feedback propagation time. The feedback solution is not purely delay insensitive but is effectively delay insensitive.

3. Two Value NULL Convention Logic™

We have shown how to make Boolean logic purely symbolically complete as a four value logic system and practically symbolically complete as a three value logic system, but three and four value logic systems are not commercially viable. The **next step** is to show how the NULL convention can be applied to a system with only two values such as our favorite digital electronic implementation environment.

If the logic is limited to two values (0 volts and 5 volts for instance) in its primitive mutually exclusive value assertion domain (a wire for instance) and one value must be assigned to express NULL (0 volts), this leaves only one value remaining to express data. This means that each wire can express only one data meaning which we will call DATA (5 volts). Two data meanings, True and False for instance, must be expressed with two wires (one wire asserting DATA means TRUE, the other wire asserting DATA means FALSE). With no inherent means to prevent any two wires from expressing their DATA values simultaneously there is no longer an inherently enforced mutually exclusive value assertion domain for data meanings as there was when two data meanings were by physical necessity mutually exclusively asserted by a single wire. Since there must be mutually exclusive assertion of DATA values it must be reestablished by convention.

Each wire is still an inherent mutually exclusive value assertion domain that asserts either DATA or NULL. To reestablish mutually exclusive assertion of data meanings, a convention that only one wire of a group of wires will assert its DATA value at a time. It is illegal or erroneous for two or more wires within this group to simultaneously assert DATA values A group of wires that assert mutually exclusive data meanings will be called a **mutually exclusive assertion group**.

A mutually exclusive assertion group can be any size. A group of ten wires can directly express decimal numbers with each wire expressing a digit value. A group of two wires can directly express True and False. The two wire mutually exclusive assertion group is identical to dual rail encoding, which is traditionally used as a transmission protocol between speed independent circuits. The mutually exclusive assertion group in NULL Convention Logic™ is a much more general concept than dual rail transmission encoding. It is not just a transmission protocol but is inherent in the logic itself.

THE SINGLE DATA VALUE LOGIC GATE

NULL expresses a control meaning and is meaningless with respect to data. In fact NULL means "not data", so the NULL value cannot be considered in resolving data value sets. Since each input to a gate can express only one DATA value (5 volts for instance) there can be no combinations of different data values as in Boolean logic. The only discriminable property available when combining wires at the input to a gate is how many DATA values are presented. Therefore, NULL Convention Logic™ gates must be **discrete threshold gates**. A complete input data set for a threshold gate is sufficient DATA values to meet its threshold. Figure 5 shows a 5 input / threshold 3 gate. If any three or more inputs are DATA the gate will assert a result DATA value. Otherwise it asserts a NULL value.

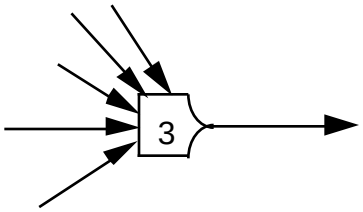

Figure 5. Threshold gate.

The most direct way to see how discrete threshold gates combine to make an effective logic circuit is via an example. The half adder will serve as the example and relate the single data value form of NULL Convention Logic™ to traditional Boolean logic. Figure 6 shows a conventional Boolean logic half adder circuit with its clock and Figure 7 shows a NULL Convention Logic™ half adder circuit.

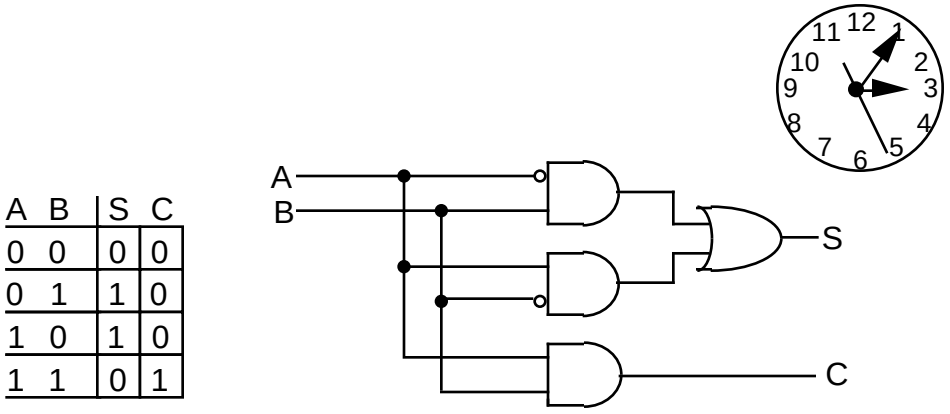

Figure 6. Boolean logic half adder circuit with clock.

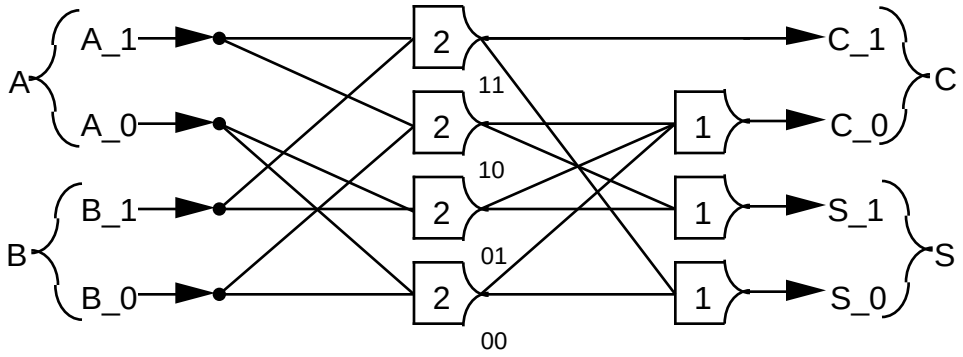

Figure 7. NULL Convention half adder circuit.

The Boolean half adder circuit expresses 4 possible input data values with two data values (0, 1) on two wires (A, B). The NULL Convention half adder circuit expresses the same 4 possible input data values with one data value (DATA) on four wires (A_0, A_1, B_0, B_1). Each binary data value A, B, C and S is expressed by a mutually exclusive assertion group of two wires (A_0, A_1 and B_0, B_1 and C_0, C_1 and S_0, S_1). Only one wire in each group can assert DATA at a time, so a complete input data set for the circuit is two DATA values, one from each group (A and B). The completeness of input criteria for each threshold 2 gate is 2 DATA values. It can be seen that for any threshold 2 gate to assert a data result value there must be one DATA value asserted in each input group and that only one threshold 2 gate at a time will assert a result data value as long as the mutually exclusive assertion convention is enforced for the input groups. **The gates, as well as the circuit as a whole, enforce the completeness of input criteria for data**. When the result values transition from NULL to a complete result data set, which in this case is one DATA value for each result group (C and S), then the asserted result values are a correct resolution of a complete input data set. The completeness of input criteria for each gate scales up for the circuit as a whole and as before, **the completeness of resolution can be determined by simply monitoring the result values**. Figure 8 shows the circuit behavior for all four possible input combinations. Bold lines are asserting DATA and thin lines are asserting NULL.

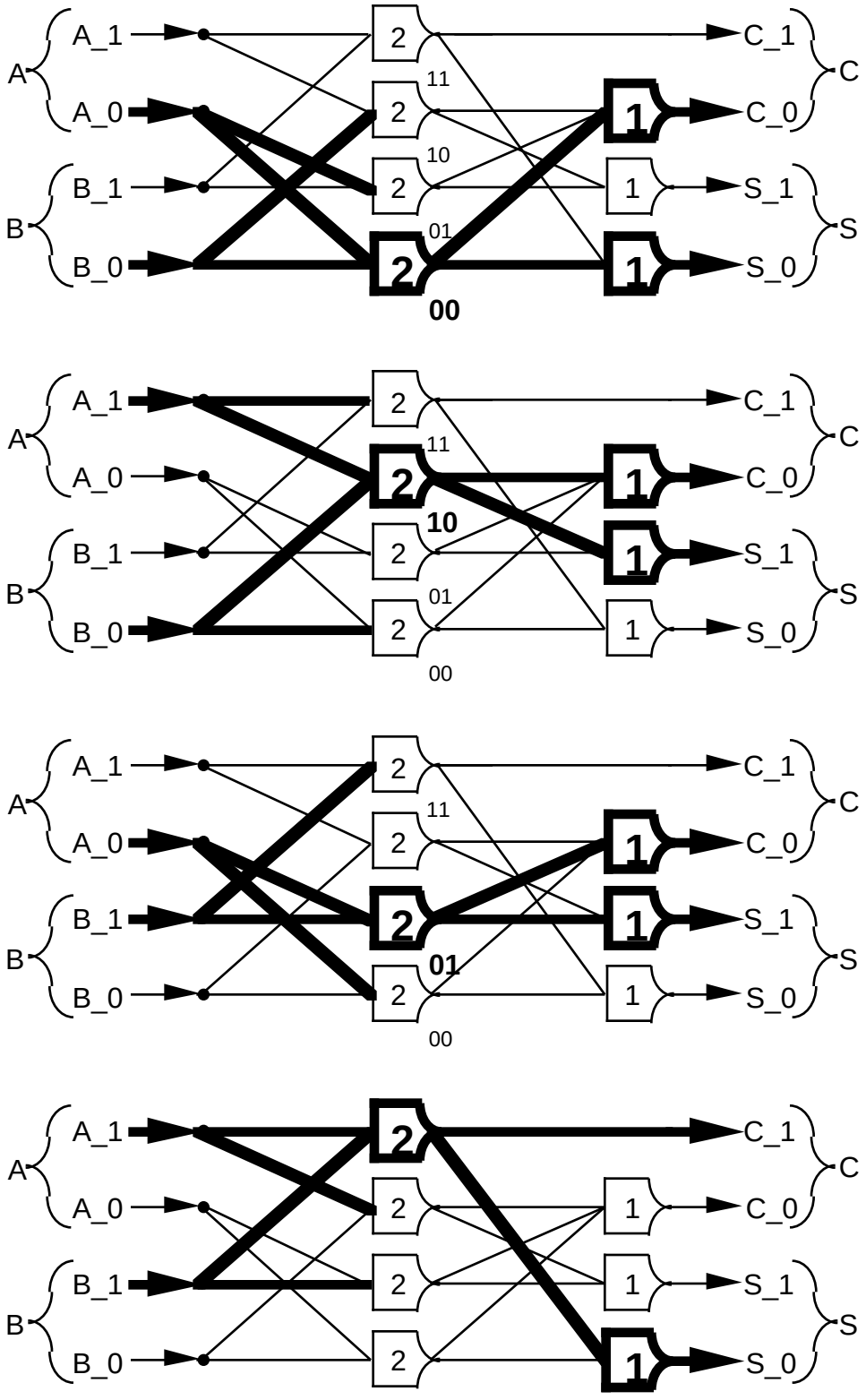

Figure 8. NULL convention half adder circuit behavior.

Another important point to make is that if the mutually exclusive assertion convention is enforced for the input of the circuit, then the circuit itself maintains the convention at its output . A system of such circuits communicating among themselves will maintain the convention among

themselves. The convention only has to be explicitly enforced at the input to the system. It is self maintaining within the system.

THE NULL WAVE FRONT AGAIN

A single data value NULL Convention Logic™ circuit still has to be returned to an all NULL state before accepting a new input data set to resolve. As can be seen from the above examples, when a gate falls below its threshold it will assert a NULL value. The result values can become all NULL while there are still DATA values lingering on the input and internal to the circuit as shown in Figure 9.

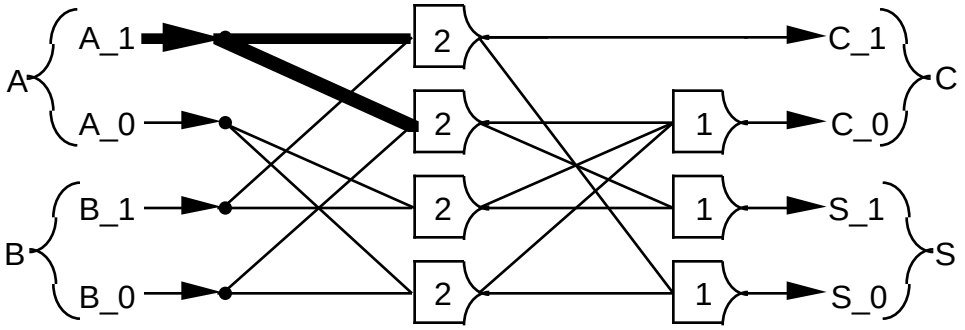

Figure 9. Lack of completeness of input criteria for NULL.

This is identical to the situation with the previous Boolean logic examples and the same two solutions apply. There is a feedback solution and an Intermediate value solution.

THE FEEDBACK SOLUTION

The feedback solution is simple, straightforward and inexpensive. To provide the necessary hysteresis behavior, the result value is fed back with a weight of one less than the threshold as shown in Figure 10. The rounded gate symbol will represent a gate with the hysteresis behavior. This hysteresis behavior may be provided by feedback internal to the gate or by some inherent behavior of the gate implementation approach.

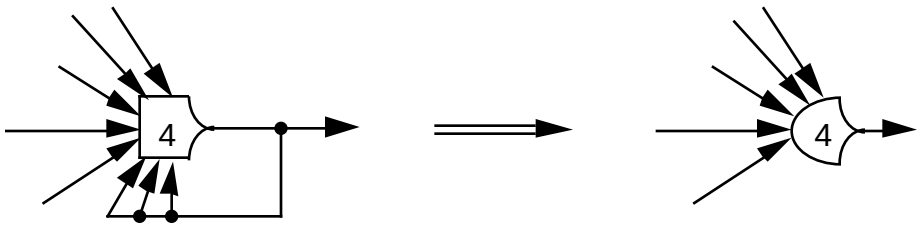

Figure 10. Threshold gate with weighted feedback of threshold -1.

Beginning in a NULL state with all inputs NULL and asserting a NULL result value, the gate will not assert a DATA result value until its input data set is complete, which in this case is four DATAs. With the feedback weight of three, the gate will not transition to a NULL result value until all of its input values are NULL. As long as at least one input value remains DATA, the gate meets its threshold and will continue asserting a DATA result value. It does not fall below its threshold and transition to NULL output until all inputs are NULL. This behavior is shown in Figure 11.

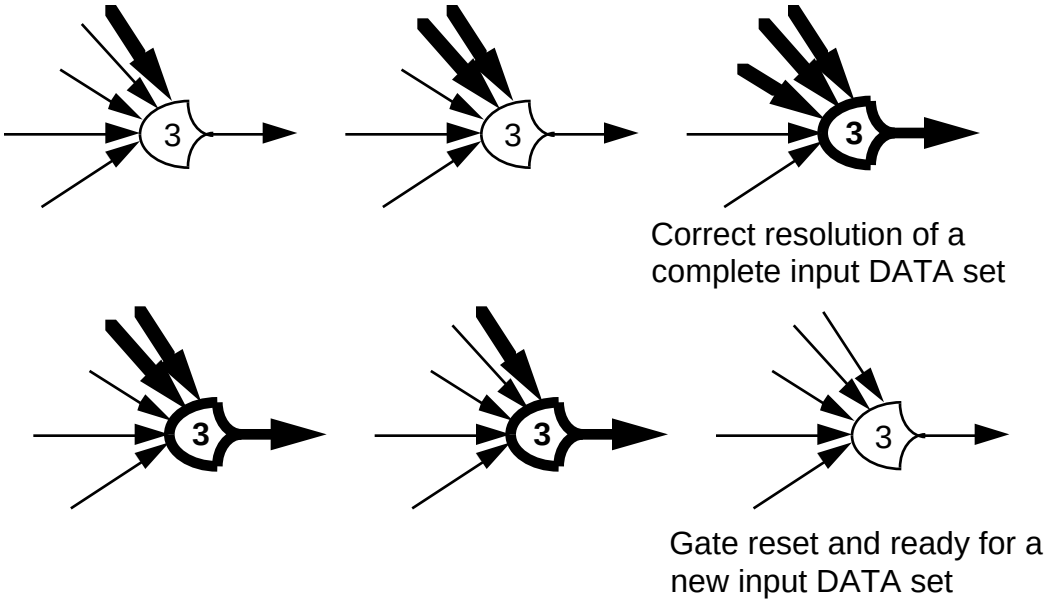

Figure 11. Hysteresis behavior of NULL Convention Logic™ gate.

The gate enforces the completeness of input criteria for both data in relation to NULL and for NULL in relation to DATA. A threshold 1 gate does not require the feedback connection because its completeness of input criteria is inherently enforced for both DATA and NULL.

THE ENCODED INTERMEDIATE VALUE SOLUTION

Since we are limited to two values in the most primitive mutually exclusive value assertion domain we can't just add another value as we did in the Boolean logic example, so we have to encode another value. The intermediate value solution involves another level of encoding convention on top of the mutually exclusive assertion group encoding. As things stand so far, each data meaning is expressed with one data value on one wire. True is one wire and False is another wire. Now we will express each data meaning with two wires. True will be expressed with two wires and False will be expressed with two wires. encoded_DATA, encoded_INTERMEDIATE and encoded_NULL are expressed on these two wires with the following encoding.

DATA, DATA -> encoded_DATA
DATA, NULL -> encoded_INTERMEDIATE
NULL, DATA -> encoded_INTERMEDIATE
NULL, NULL -> encoded_NULL

As shown in Figure 12, each input to a gate is two wires and a single intermediate value gate consists of several threshold gates without hysteresis.

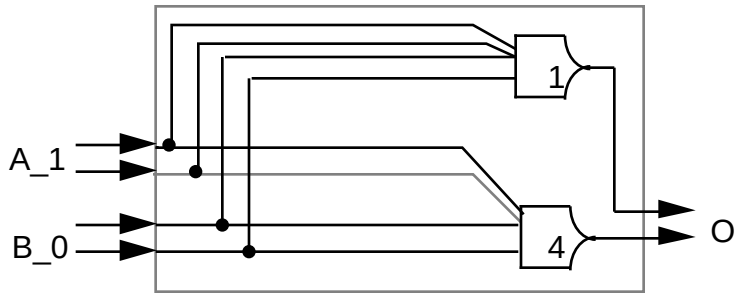

Figure 12. Intermediate encoded 2 input threshold 2 gate.

Starting with the intermediate value gate in an all NULL state with encoded_NULL values on the inputs and asserting an encoded_NULL value. If one DATA value is applied to A_1 the threshold 1 gate meets its threshold and will assert a DATA result value and the result value for the composite gate will assert DATA, NULL to express the encoded_INTERMEDIATE value. The result value remains encoded_INTERMEDIATE until all four input wires are DATA, which means that both inputs are presenting encoded_DATA values and there is a complete input data set, at which time the threshold 4 gate asserts a DATA value and the intermediate value gate asserts DATA, DATA to express the encoded_DATA value. As soon as one input wire becomes NULL the threshold 4 gate falls below its threshold and asserts NULL and the asserted result for the intermediate value gate is DATA, NULL expressing the encoded_INTERMEDIATE value. The encoded_INTERMEDIATE value is maintained until all the input wires become NULL at which time the threshold 1 gate asserts a NULL value and the result value for the composite gate is NULL, NULL expressing the encoded_NULL value.

The Intermediate value composite gate asserts a encoded_NULL code only when all of its input values are NULL and expresses a encoded_DATA only when it has a complete set of input data values. **It enforces the completeness of input criteria for both data and NULL.** Figure 13 shows a three input threshold two gate.

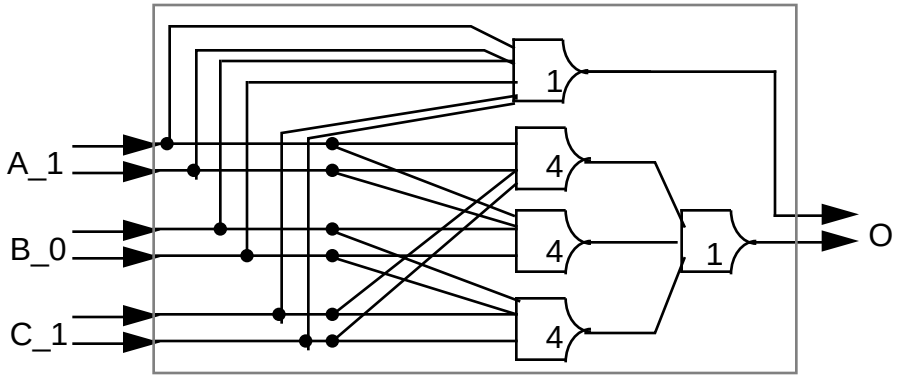

Figure 13. Intermediate encoded 3 input threshold 2 gate.

Figure 14 shows the Intermediate encoded Half adder circuit.

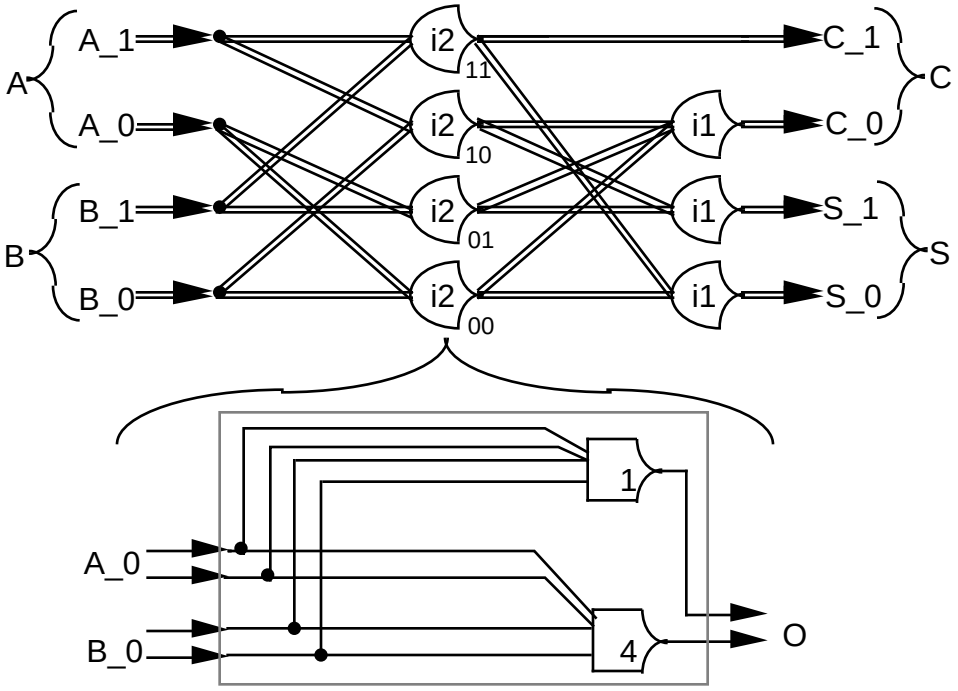

Figure 14. Intermediate encoded half adder circuit.

Translating between intermediate encoded gates and hysteresis gates is straightforward. As shown in Figure 15 the translation from hysteresis to encoded is simply a matter of fanning out the single wire. Translating from encoded gates to hysteresis gates can be done with a single threshold 2 hysteresis gate which discriminates encoded_DATA from encoded_NULL and ignores encoded_INTERMEDIATE.

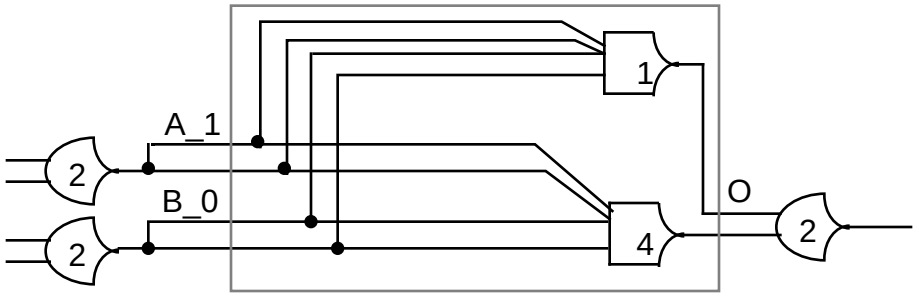

Figure 15. Translating between intermediate encoded gates and hysteresis gates.

As can be seen Intermediate encoding is very expensive in terms of resources. The wires are all doubled and each encoded gate is several simple threshold gates. The feedback solution is more practical and economical.

Now that both forms of NULL Convention threshold gates enforce the completeness of input criteria for both DATA in relation to NULL and for NULL in relation to DATA, their behavior again scales up for circuits as a whole. Figure 16 shows the data-NULL cycle for a single data value NULL Convention Logic™ circuit.

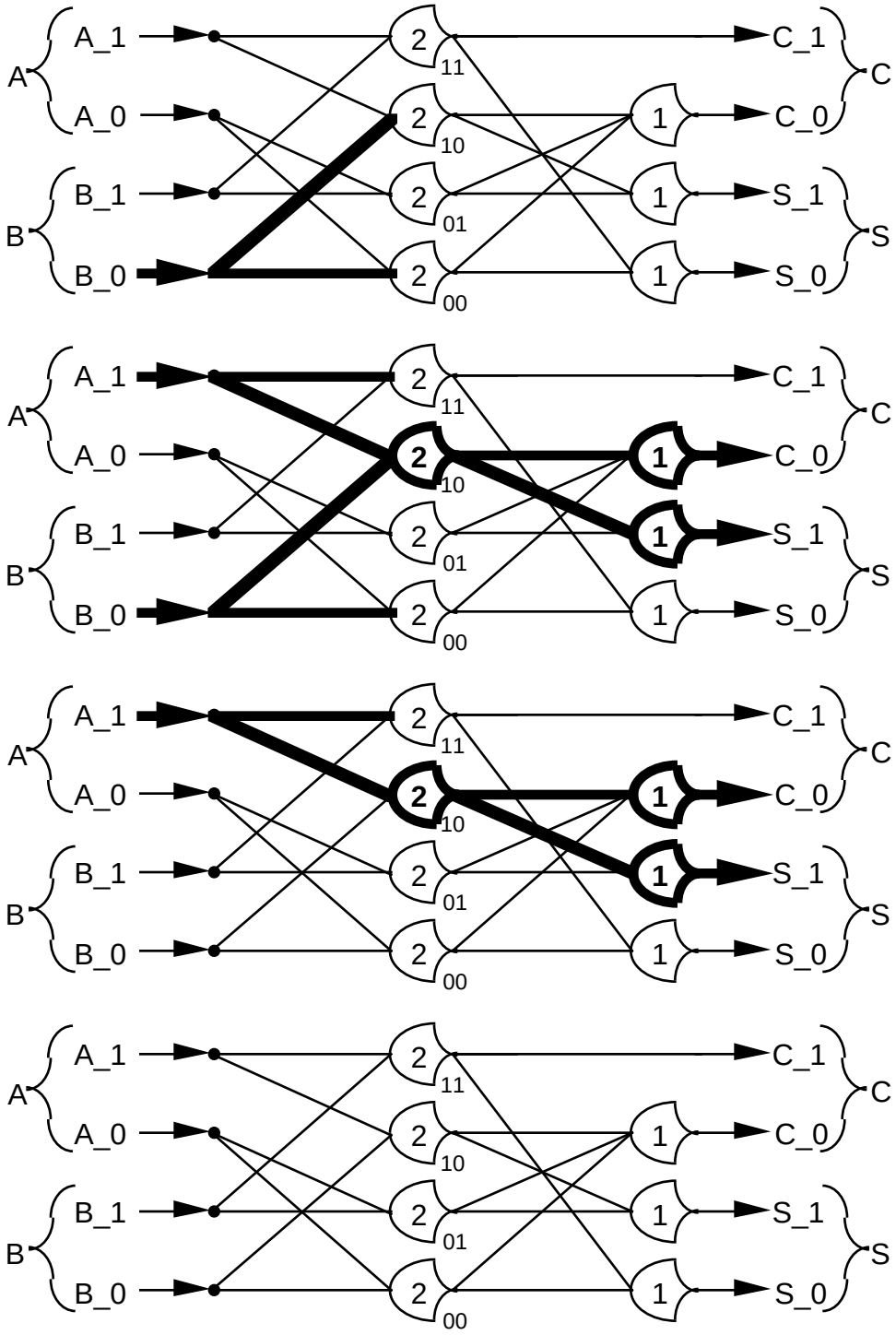

Figure 16. NULL-DATA cycle for hysteresis gate circuit.

Beginning with the circuit in an all NULL state one DATA value is applied in one of the groups. The result values remain all NULL since no threshold is met. When a DATA value is applied in the second group, a complete input data set is present, the threshold of a gate is met which asserts a DATA value which propagates to the result values. One result value in each output group asserts a DATA value which constitutes a complete result data set and expresses the correct

resolution of a complete input data set. The circuit as a whole enforces the completeness of input criteria for DATA in relation to NULL and only asserts a complete result data set when a complete input data set is presented to the circuit.

One of the input DATA values becomes NULL, but the threshold 2 gate and the circuit continue asserting DATA result values. Only when all inputs to the circuit are NULL does the threshold 2 gate and hence the circuit transition their result values to NULL. The circuit as a whole enforces the completeness of input criteria for NULL in relation to DATA and only asserts all NULL result values when the input to the circuit is all NULL and the NULL values have propagated through the circuit.

As with the Boolean logic examples with three and four value logic, the completion of resolution of a complete input data set and the readiness of the circuit to receive a new input data set to resolve can be determined by simply monitoring the result values. **The circuit is symbolically complete and manages quite on its own its interactions with the rest of the world**.

INTERESTING OBSERVATIONS ON SINGLE DATA VALUE NULL CONVENTION LOGIC™ CIRCUITS

A single data value NULL Convention Logic™ circuit can be conveniently monitored for faults in such a way that it cannot tell a lie as shown in figure 17.

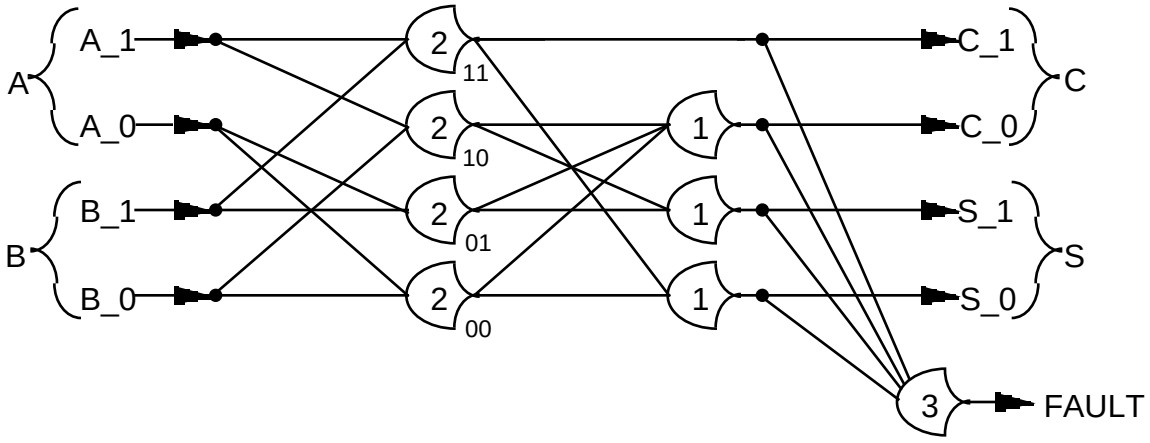

Figure 17. Fault monitoring NULL Convention Logic™ circuit.

If exactly one DATA value is asserted in each output group it is the correct resolution of a complete input data set. If there are ever three DATA result values simultaneously asserted, it is an error and the threshold three gate will announce the error. If the circuit only asserts one DATA result value it will fail to announce completion of resolution and this can be detected with a watchdog timer. **A NULL Convention Logic™ circuit will always either:**

1. Assert a correct result.
2. Fail to complete resolution.
3. Assert an explicit error signal.

A fault will be detected as soon as it causes an actual resolution error. This holds for all single point faults. It is possible for a NULL Convention Logic™ circuit to tell a lie but it requires two coordinated faults that produce a valid encoding which is, nevertheless, erroneous.

Single data value NULL Convention Logic™ circuits are similar to neural nets as shown in figure 18.

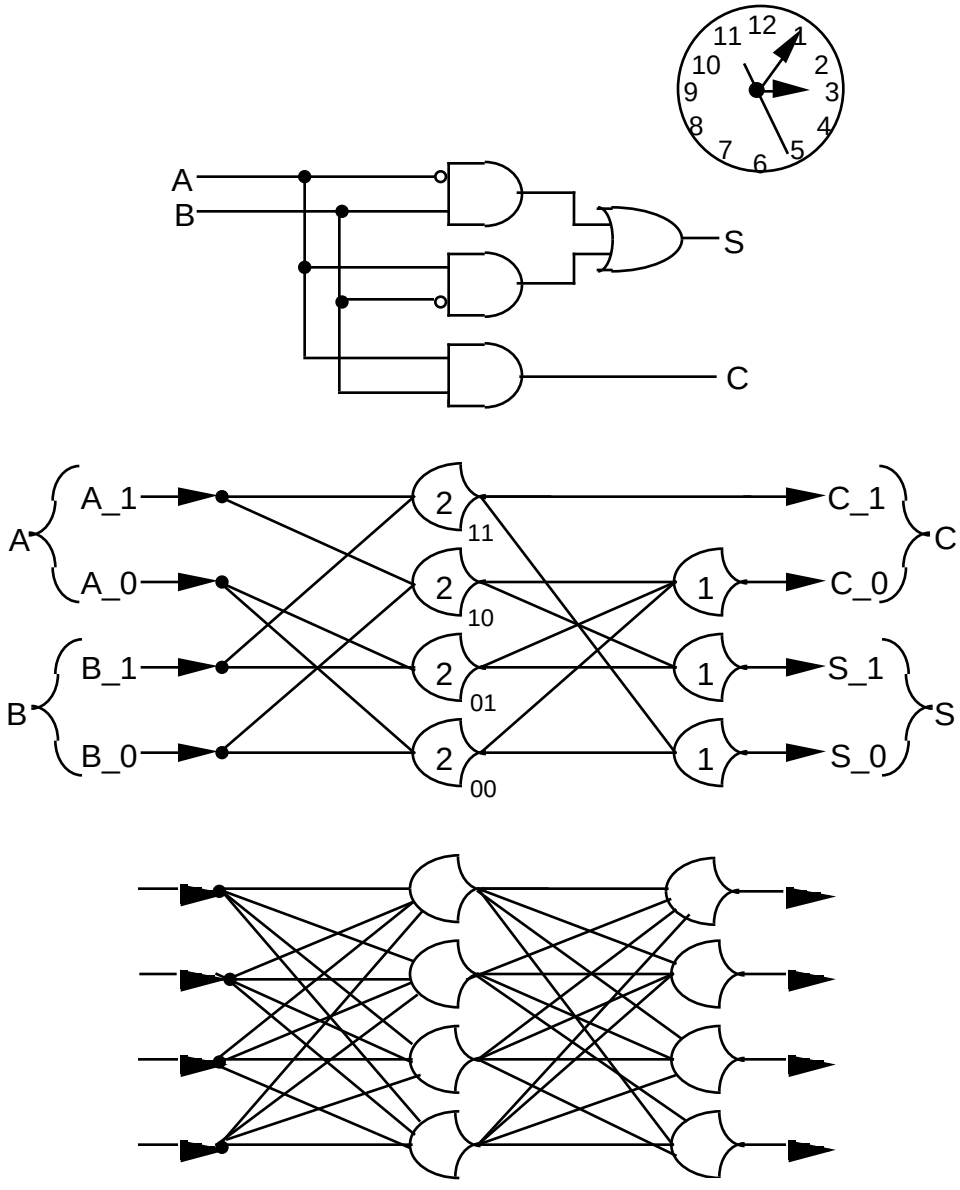

Figure 18. Similarity to neural nets.

The single data value NULL Convention Logic™ circuit in the middle performs the same discrete symbolic processing as the Boolean circuit at the top, while at the same time being a more complete and autonomous expression of the process than the Boolean logic circuit. The fully connected neural net at the bottom, with an input layer, a hidden layer and an output layer can be configured to identically match the NULL Convention Logic™ circuit by setting the thresholds of each node and setting the weights of each connection appropriately to zero or one. The NULL Convention Logic™ circuit might be viewed as a pretrained neural net.

4. The Asynchronous Register

To build a system out of NULL Convention combinational logic circuits we must be able to manage the communication and interaction among its component circuits. There must be an asynchronous register, as shown in figure 19, that monitors completeness of resolution as well as readiness to accept new input data sets and stores the complete set of data values and the all NULL values between circuits. Each combinational circuit will have an asynchronous register at its input.

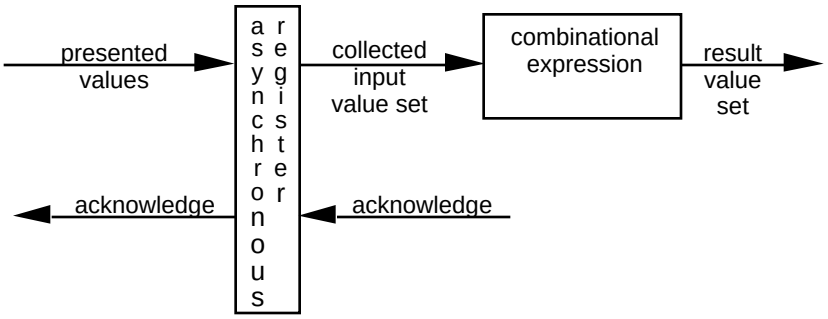

Figure 19. The asynchronous register.

The asynchronous register shown in Figure 20 is simply a rank of NULL Convention Logic™ threshold gates with feedback hysteresis (Intermediate value gates won't work here) and a single gate that watches for complete data sets and all NULL states. It will manage the interaction among combinational circuits and **is itself a very simple NULL Convention Logic™ circuit**.

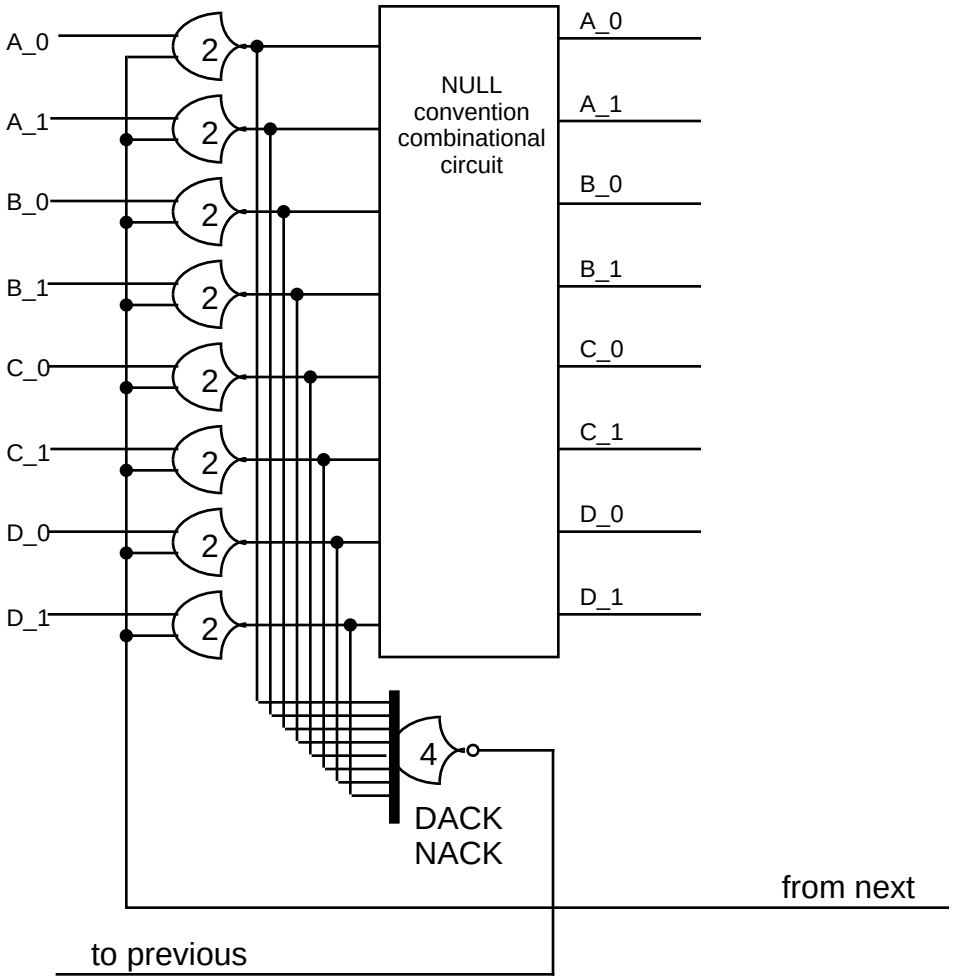

Figure 20. The NULL Convention Logic™ asynchronous register.

Each register gate (the threshold 2 gates) receives one wire with a data meaning and one wire with a control meaning. The threshold 4 gate is the watcher gate. When it sees a complete set of DATA values it will assert DATA and when it sees all NULL values it transitions to asserting NULL. When the watcher gate sees a complete data set, which in this case is one DATA value each for A, B, C and D input groups, it means that a complete data set has been received and stored by the register gates and it will transition its result value to DATA. The watcher gate will continue to assert the DATA value until all of its input values are NULL, which means that the register gates have received and stored all NULL values.

The control input for each register gate comes from the watcher of the next or downstream register. When the next register has received and stored a DATA wave front playing through the combinational circuit, it sends back a DATA acknowledge (DACK) that says "I have received and stored the DATA wave front and now you can send the NULL wave front". This is conveyed by transitioning the control line value to NULL. But the watcher transitions its result to DATA when it sees a complete data set so an inverter is needed. The same is true for the reception of the NULL wave front. When the watcher sees all NULL values it transitions its result value to NULL. This means "I have received and stored the NULL wave front and now you can send a DATA wave front". But to authorize a DATA wave front the control value must be DATA, so again the signal must be inverted.

The control line requests the sending of a DATA wave front or a NULL wave front. The register gates are feedback hysteresis gates that enforce the completeness of input for both DATA and NULL. To pass DATA values, both inputs of a gate must be DATA. Therefore a DATA wave front cannot pass the register gates until the control line is DATA, or in other words until the next register has requested the data to be passed. When the control line becomes DATA, the DATA values presented to the register gates will be passed and as long as the control line remains DATA the data will be stored by the register gates. When a complete DATA set has been passed and stored by the register gates the watcher gate will detect the complete DATA set and transition the control line to the previous register to NULL, indicating that it has received and stored a DATA wave front and the previous register can pass a NULL wave front.

Because of the hysteresis behavior of the register gates, they will not pass a NULL wave front until both inputs are NULL. So when the NULL wave front arrives at the register gates and the control line becomes NULL, the NULL values will be passed and stored as long as the control line is remains NULL. After the NULL values are passed and stored, the watcher will detect the all NULL value set and transition the control line to the previous register to DATA indicating that it has received and stored the NULL wave front and the previous register can pass a DATA wave front.

The behavior of the asynchronous register is detailed in figure 21.

Synchronization Register Behavior

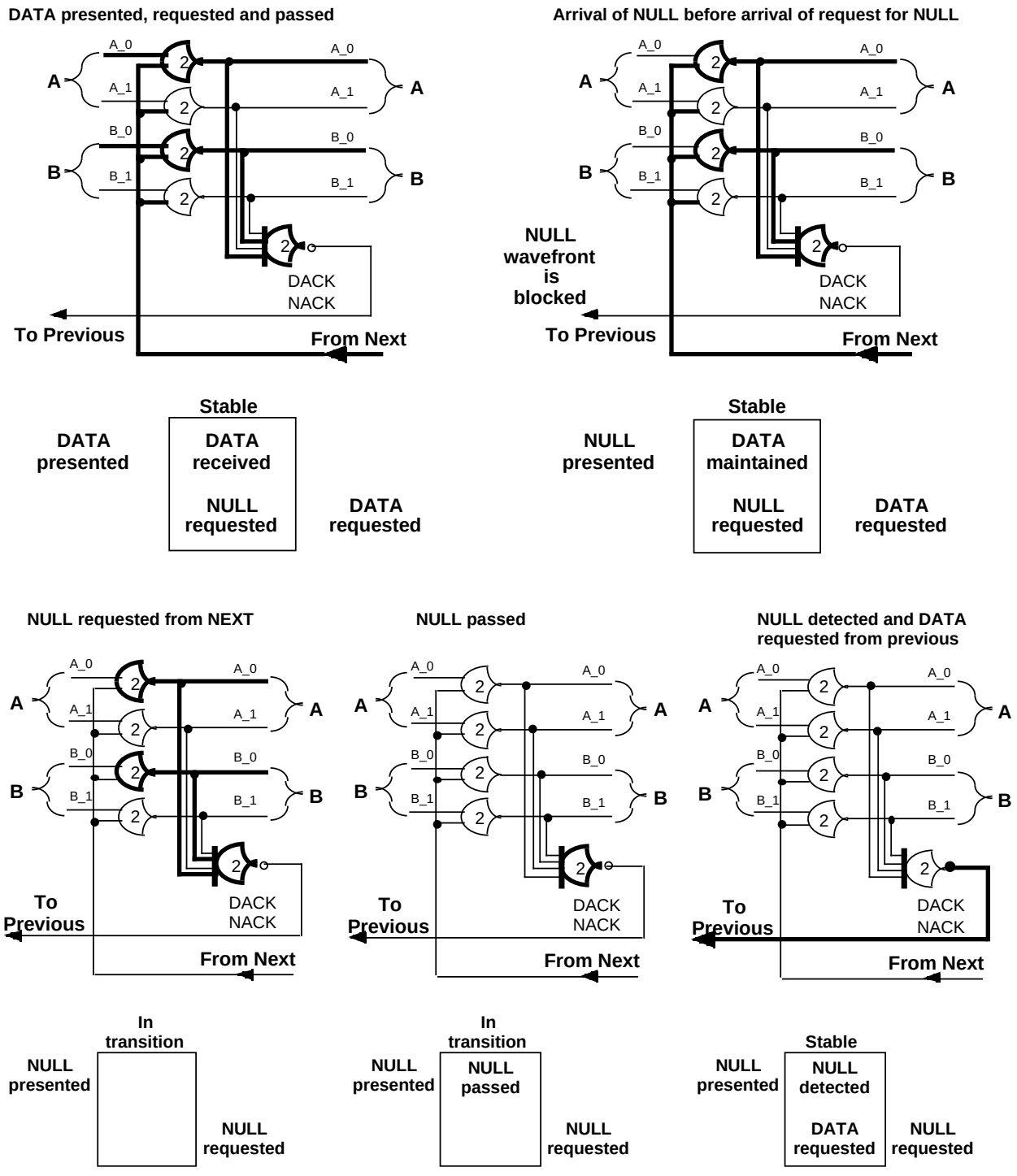

Figure 21a. Asynchronous register behavior.

Synchronization Register Behavior

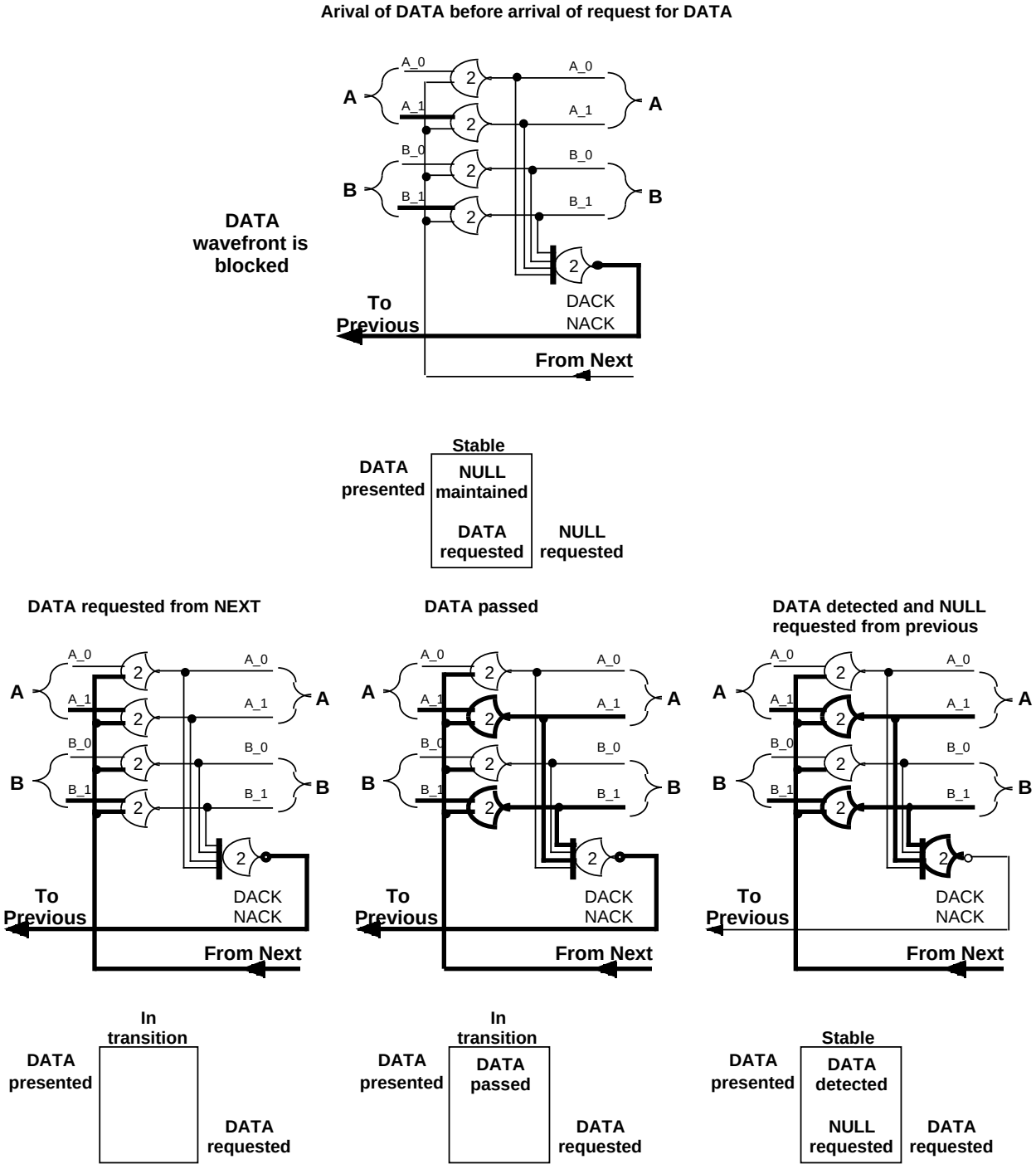

Figure 21b. Asynchronous register behavior.

Synchronization Register Behavior

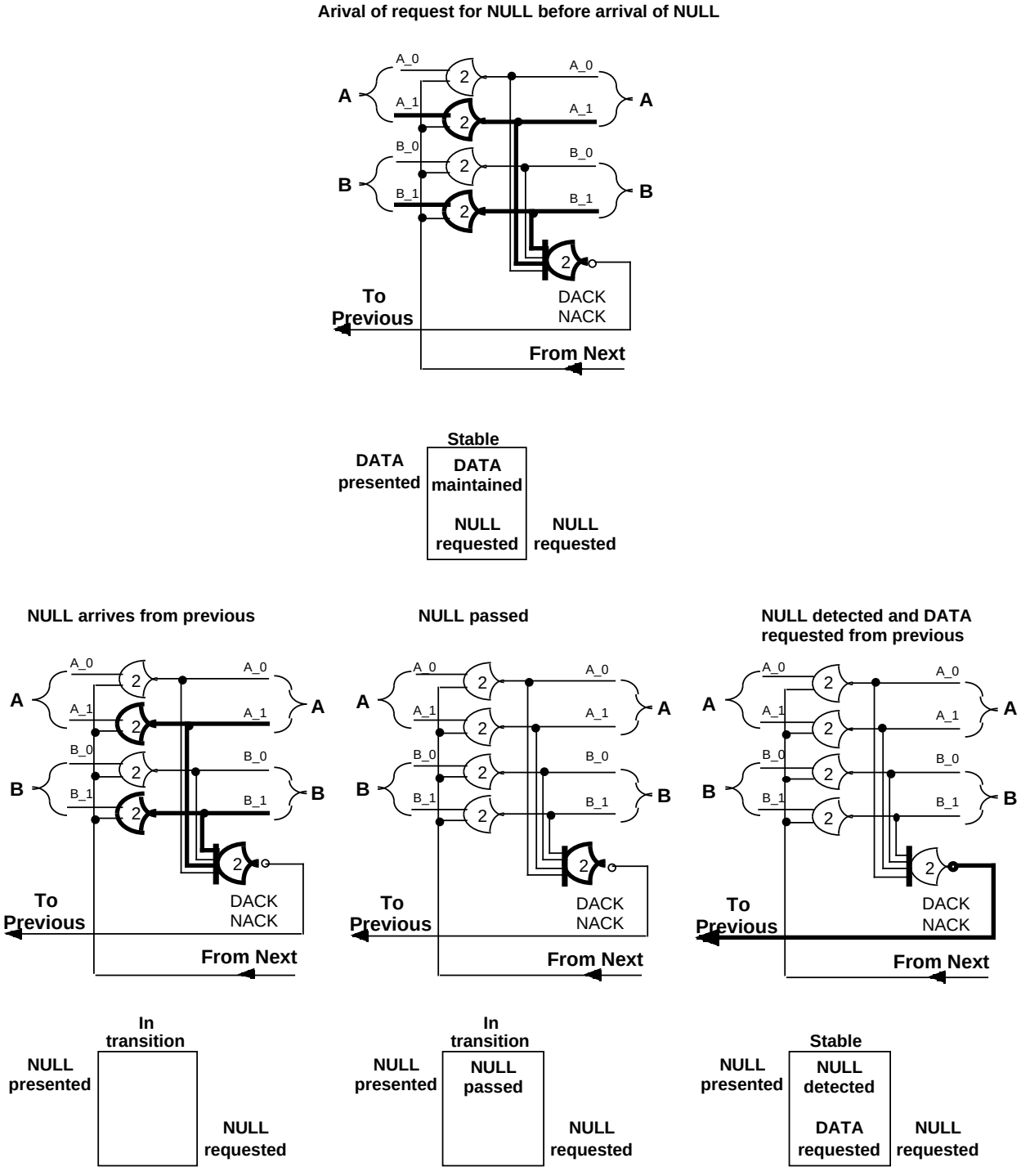

Figure 21c. Asynchronous register behavior.

Synchronization Register Behavior

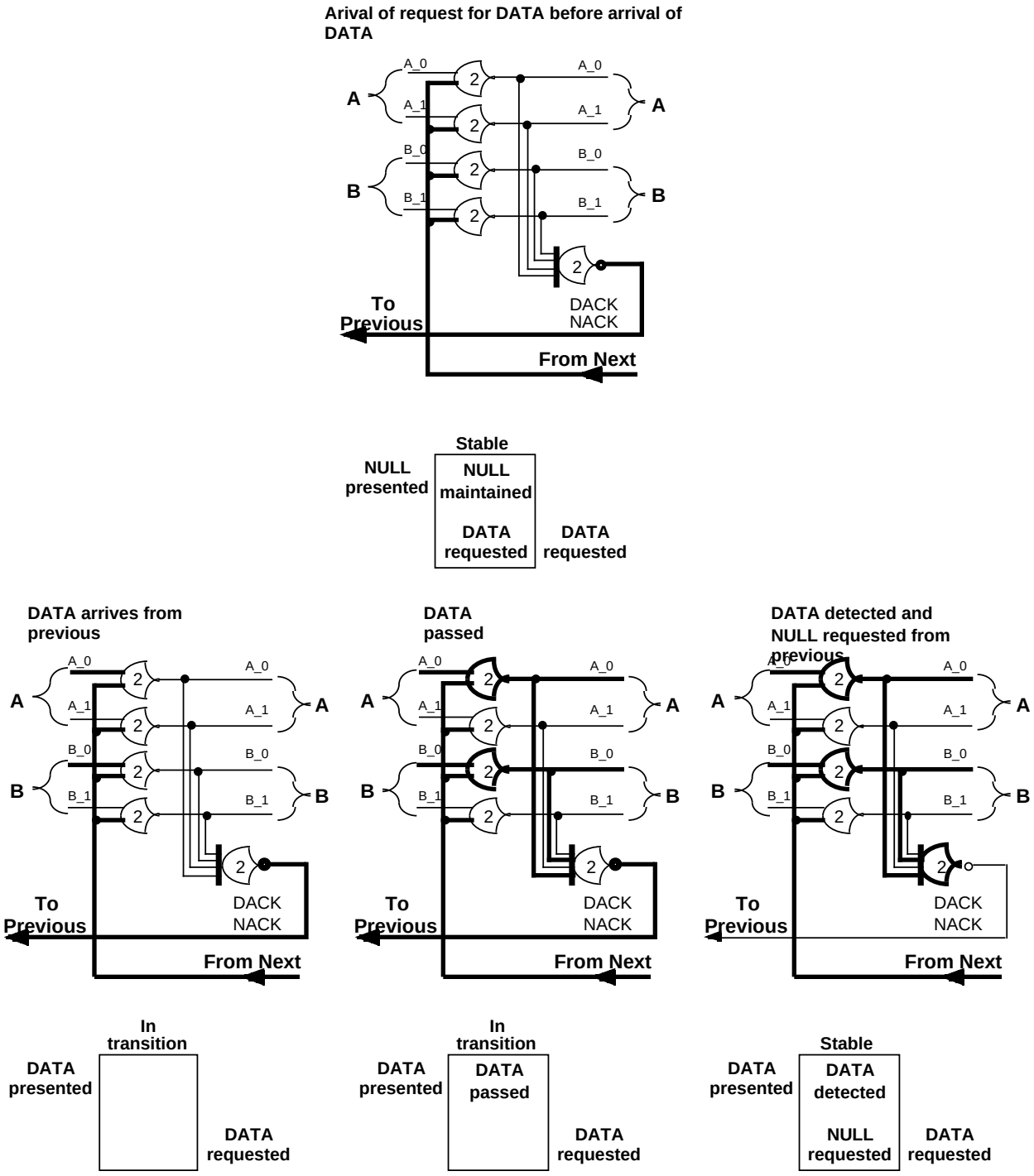

Figure 21d. Asynchronous register behavior.

ASYNCHRONOUS REGISTER SUMMARY

The asynchronous register manages the fully asynchronous flow of data wave fronts and NULL wave fronts among NULL Convention Logic™ circuits. Passage of DATA or NULL through a register occurs strictly after the request for DATA or NULL, and the detection of complete DATA or all NULL state occurs strictly after the passage and storage of the DATA or NULL values. The asynchronous registers and their behavior are fully delay insensitive and are purely symbolically determined. There are no time dependency relationships whatever. With the asynchronous register a fully delay insensitive and purely symbolically determined system can be built.

STRUCTURES OF ASYNCHRONOUS REGISTERS

The interaction behavior among combinational circuits can be seen with the simple pipeline example shown in Figure 22.

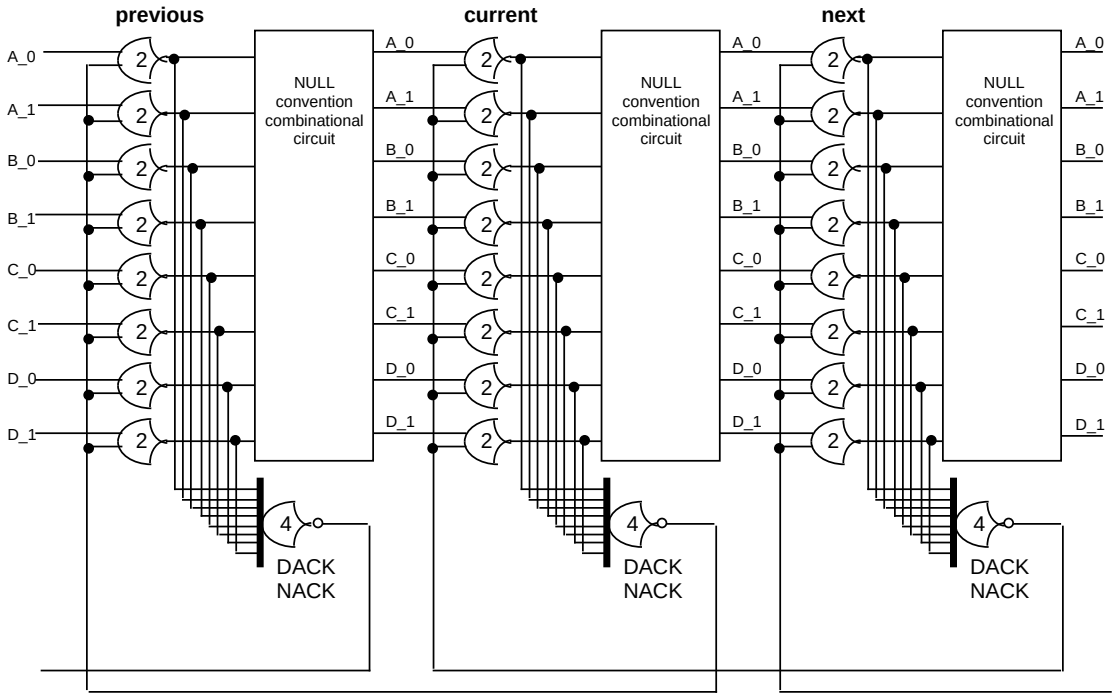

Figure 22. NULL Convention asynchronous pipeline.

Assume that all the circuits are in a NULL state and that the current watcher and the next watcher is requesting a DATA wave front and that the previous register is presenting a complete DATA set to its combinational circuit. As the wave front propagates through the previous circuit to the current register, the current register passes the data since its control line is DATA. When a complete data set is recognized by the current watcher, the current watcher transitions its control line to the previous register to NULL to indicate that it has received and stored the data wave front and the previous register can pass a NULL wave front. The requested NULL wave front from the

previous register can arrive at the current register but, as long as its control line is DATA, the NULL wave front will be blocked and the current register will maintain the presentation of the set of DATA values to the current circuit. The control line for the current register will remain DATA until the DATA wave front has propagated through the current circuit and has been received by the next register. When the next register receives and stores the DATA wave front, the DATA set no longer needs to be maintained by the current register. The next watcher detects the complete DATA set and transitions its acknowledge line to NULL to indicate that it has the DATA wave front and the current register can allow a NULL wave front through.

The registers and circuits can be configured in more complex structures than a simple pipeline. Figure 23 shows a fan in configuration of circuits

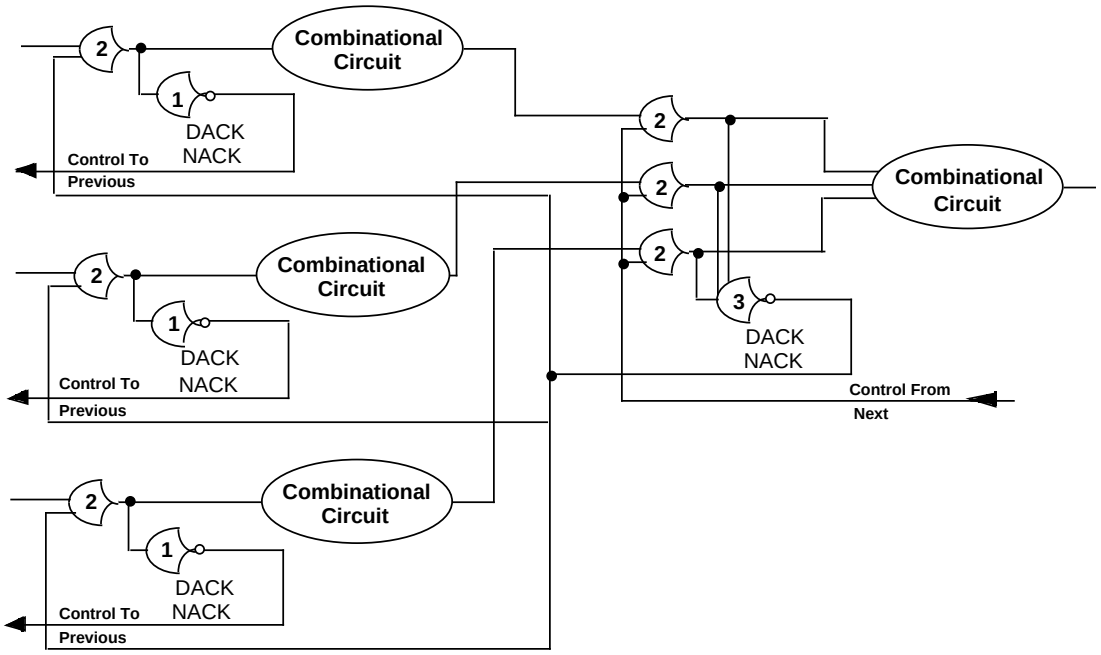

Figure 23. Fan in configuration of registers and circuits.

In Figure 23 each threshold 2 gate represents a complete rank of register gates. In this configuration three circuits present their results to a single circuit. The single circuit receives the wave fronts from all three and does not signal completeness of DATA or completeness of NULL until all three wave fronts are present and complete. It then sends the acknowledge to all three. The wave fronts of the three circuits are synchronized at the register of the single circuit.

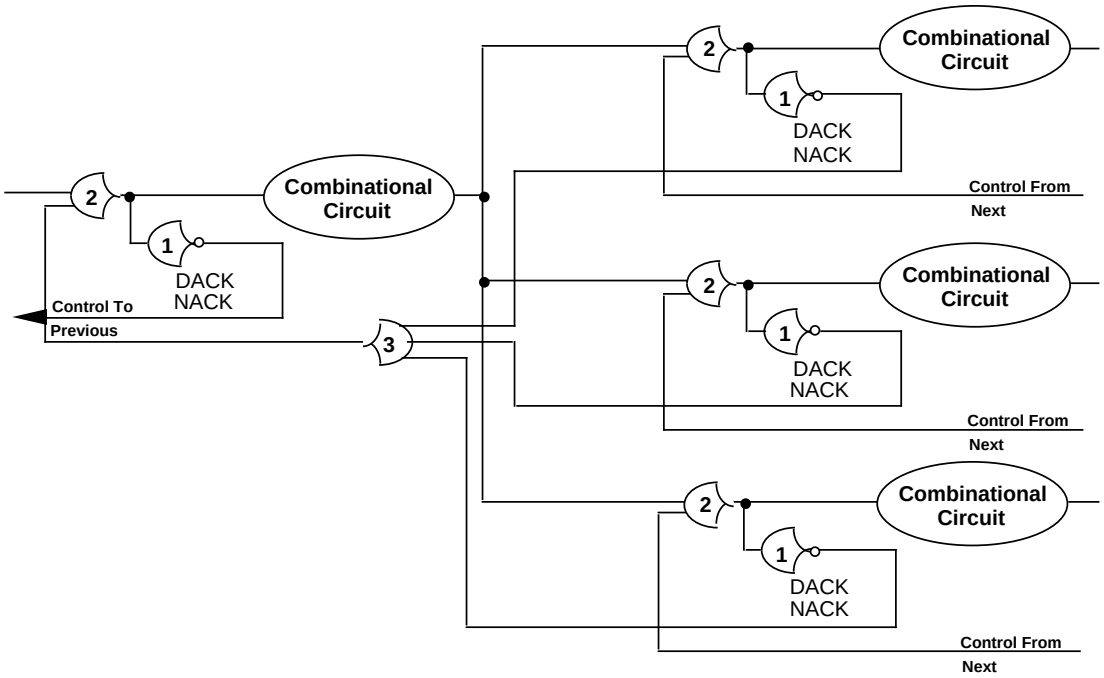

Figure 24 Fan out configuration of registers and circuits.

Figure 24 shows a fan out configuration where a single circuit delivers its result values to three circuits. Each of the three circuits receives the wave front individually and generates its own acknowledge control signal. The threshold three gate collects the acknowledge signals. It does not assert DATA until all three acknowledge inputs are DATA, meaning all three circuits have received a complete DATA wave front, and it does not assert NULL until all three acknowledge inputs are NULL, meaning that all three have received a complete NULL wave front. So the fanned out wave front is synchronized by the threshold 3 gate.

THE SEQUENTIAL CIRCUIT

The next structure required to build a system is a sequential circuit. Figure 25 shows a sequential circuit constructed with asynchronous registers and a combinational circuit.

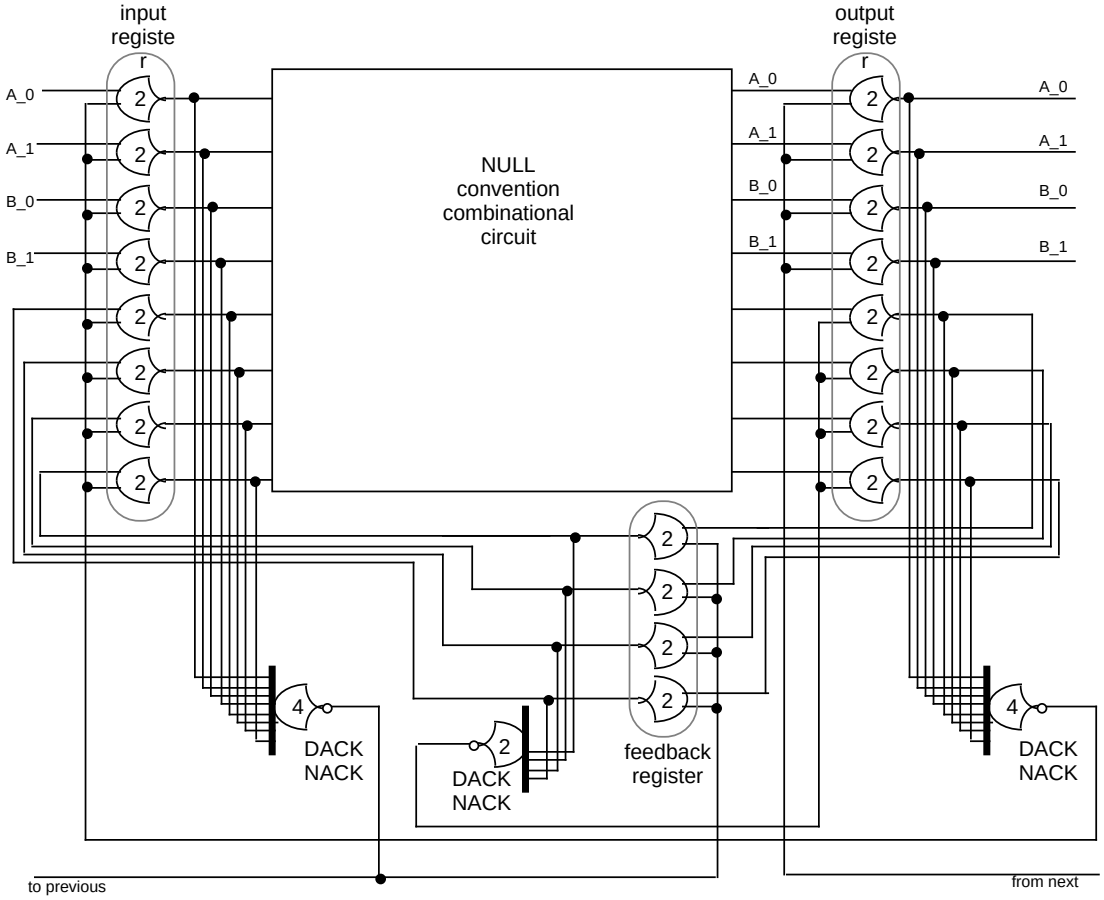

Figure 25. NULL Convention Logic™ sequential circuit.

The top four gates of the input register hold the inputs from the previous circuit. The top four gates of the output register receive the result values for the next register. The bottom four gates of the output register receive the internal state of the circuit and direct it back to the bottom four gates of the input register through the intermediate register at the bottom of the circuit which buffers successive DATA and NULL wave fronts from the output to the input of the circuit. The next DATA set arriving at the top four gates is synchronized with the previous DATA set's output arriving at the bottom four input gates from the feedback register.

The sequential circuit is symbolically complete. The behavior of the circuit is purely symbolically determined, the behavior of the asynchronous registers are purely symbolically determined and their combination into a system structure is purely symbolically determined. There is no external control authority. The circuit's behavior is driven completely by the DATA-NULL input wave fronts and acknowledge signal interaction with its predecessor circuit and with its successor circuit.

The basic elements to build a system (combinational circuits, registers and sequential circuits) have been demonstrated.

5. Some Adder Circuits

Next we show some adder circuits to demonstrate some advantages of NCL™ over previous approaches to asynchronous circuits. To give a sense of the logic we begin with the Boolean logic Minterm form of the full adder shown in figure 26.

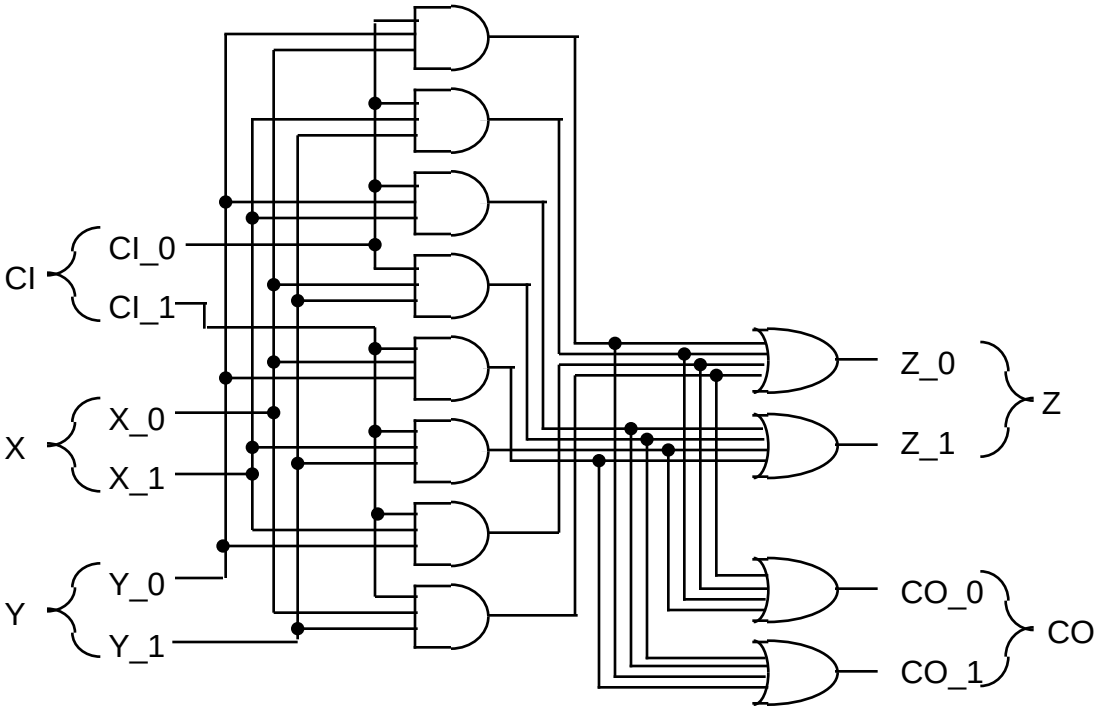

Figure 26. Minterm version of full adder.

This can be transformed directly into an asynchronous delay insensitive circuit by replacing the AND gates with 3 input C-elements as shown in figure 27. This is a well known asynchronous circuit called delay insensitive minterm synthesis or DIMS (8). What is not known is how to optimize this circuit which is expressed partly in Boolean logic and partly in terms of C-elements.

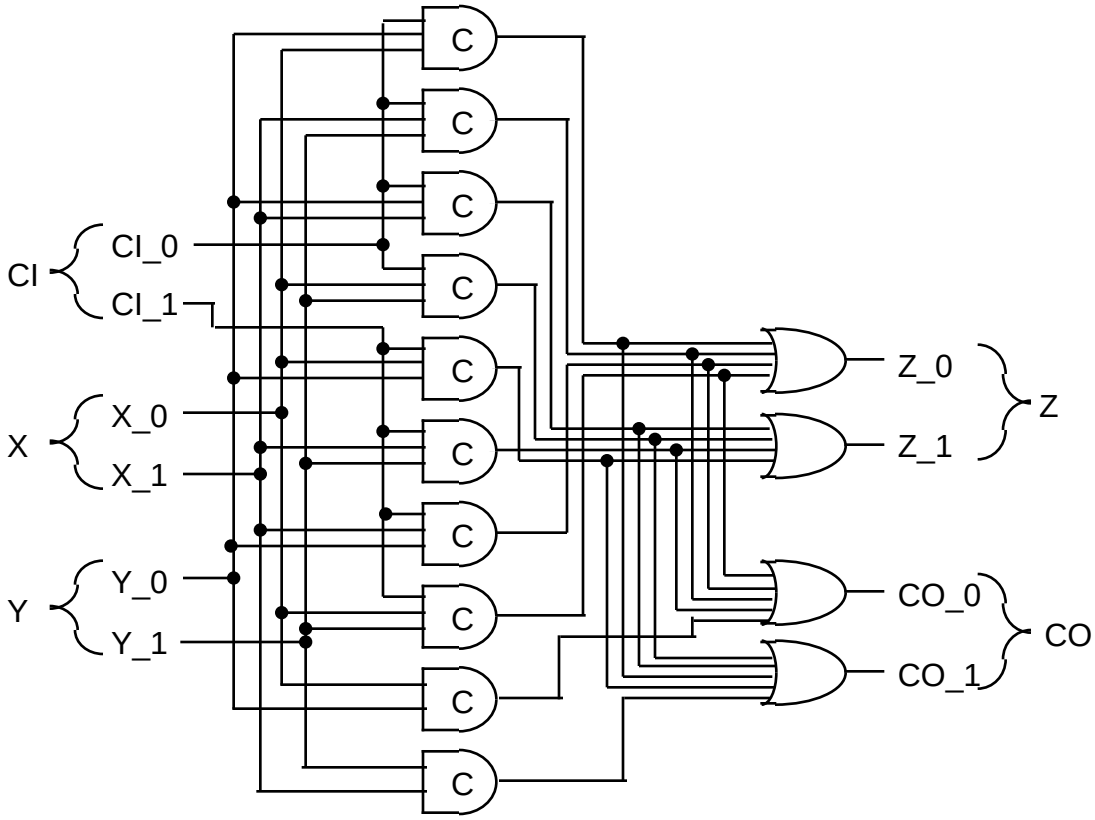

Figure 27. Minterm version of full adder.

Because NCL™ is a uniform and consistent logic, the NCL™ version of the same DIMS circuit can be optimized. Figure 28 is the NCL™ version of the DIMS full adder.

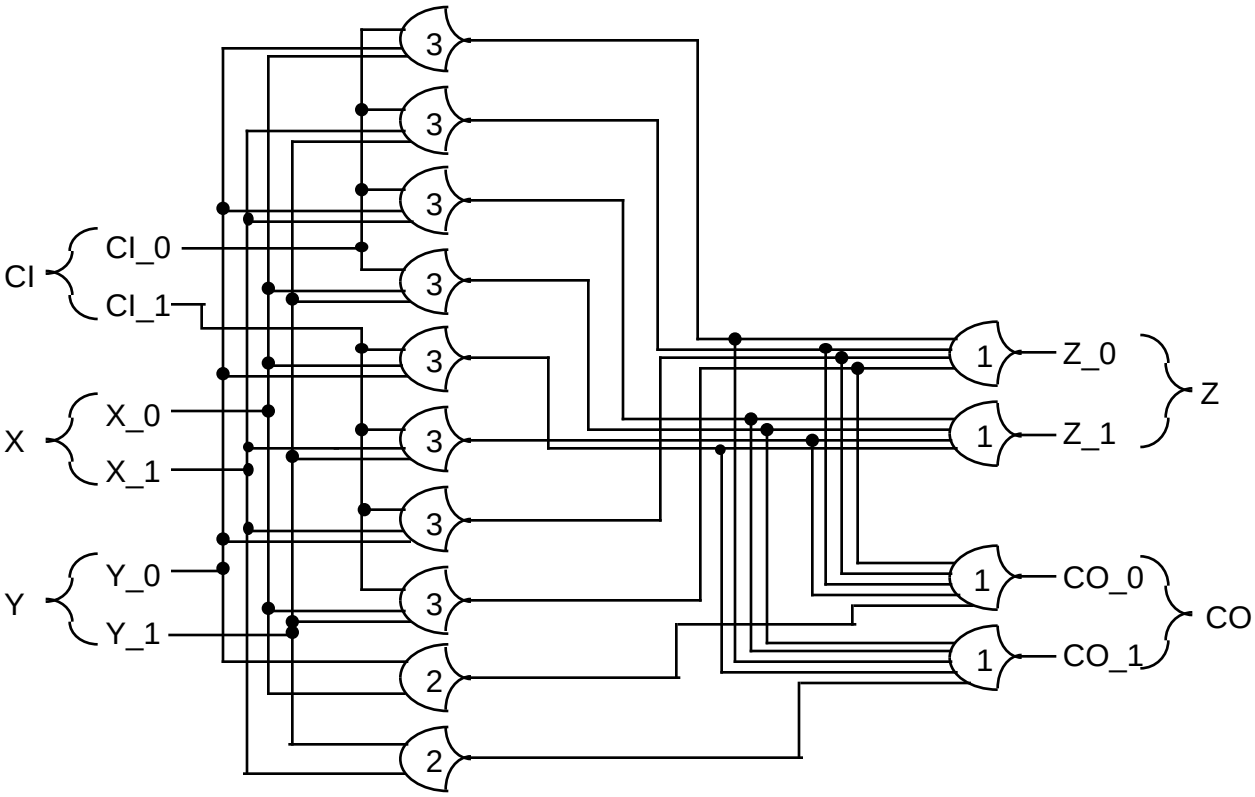

Figure 28. Minterm version of full adder.

The DIMS full adder counterpart can be optimized to the full adder circuit shown in figure 29 with expression transforms entirely within the context of NCL™.

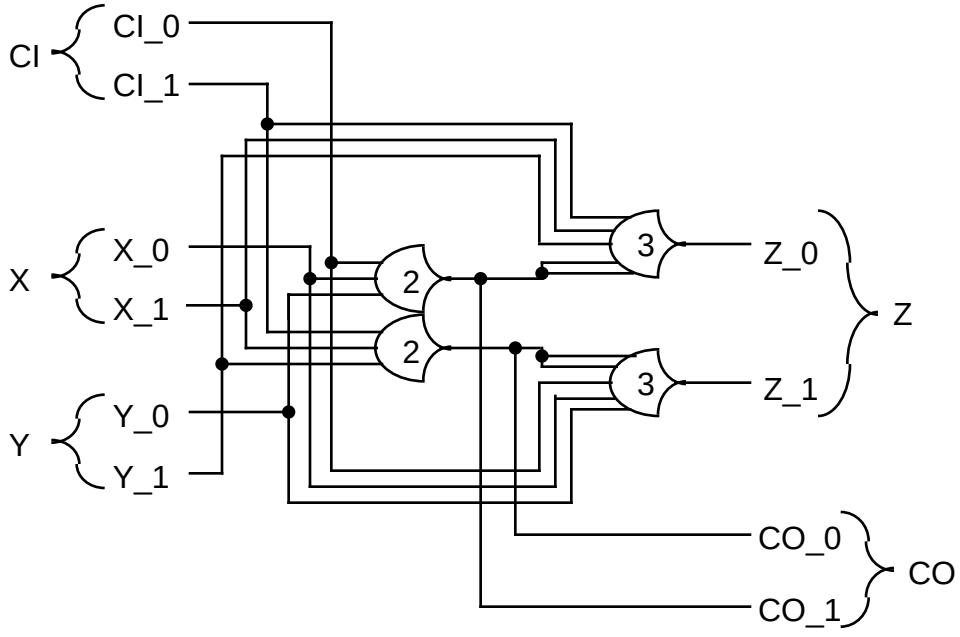

Figure 29. The optimized full adder.

Full adder circuits can be combined in the evident way to form a ripple carry adder as shown in figure 30.

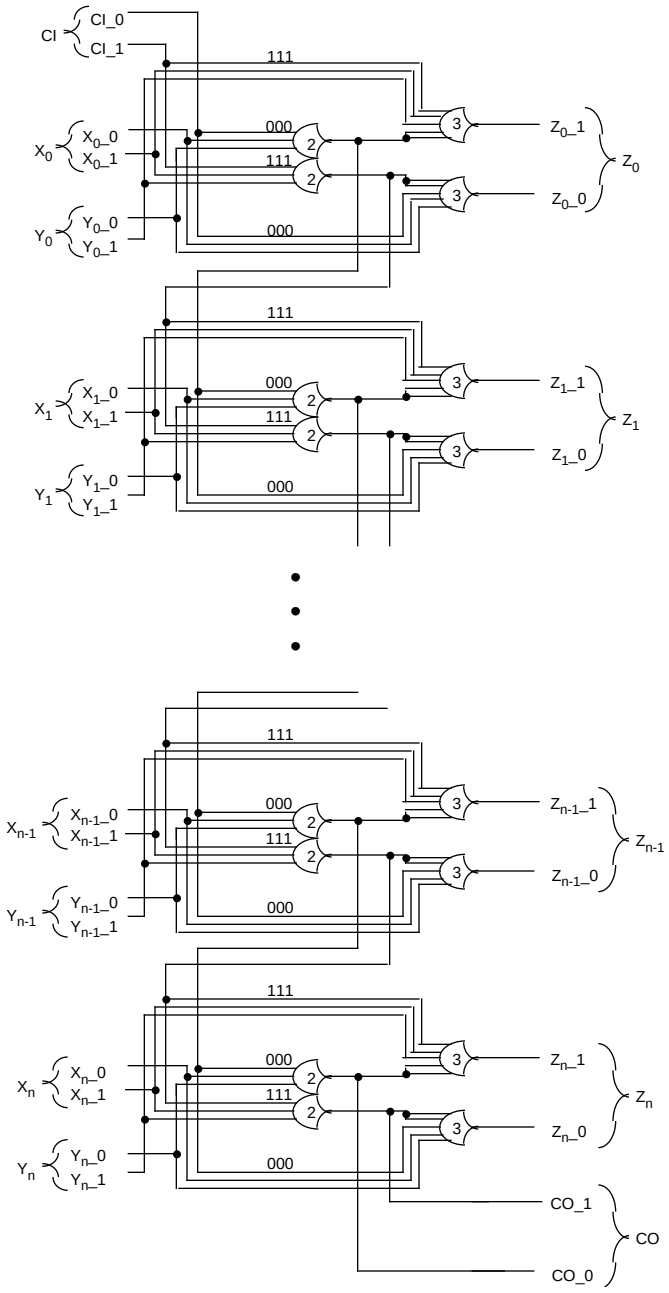

Figure 30. Ripple carry adder circuit.

Figure 31 shows the ripple carry adder with an output asynchronous register.

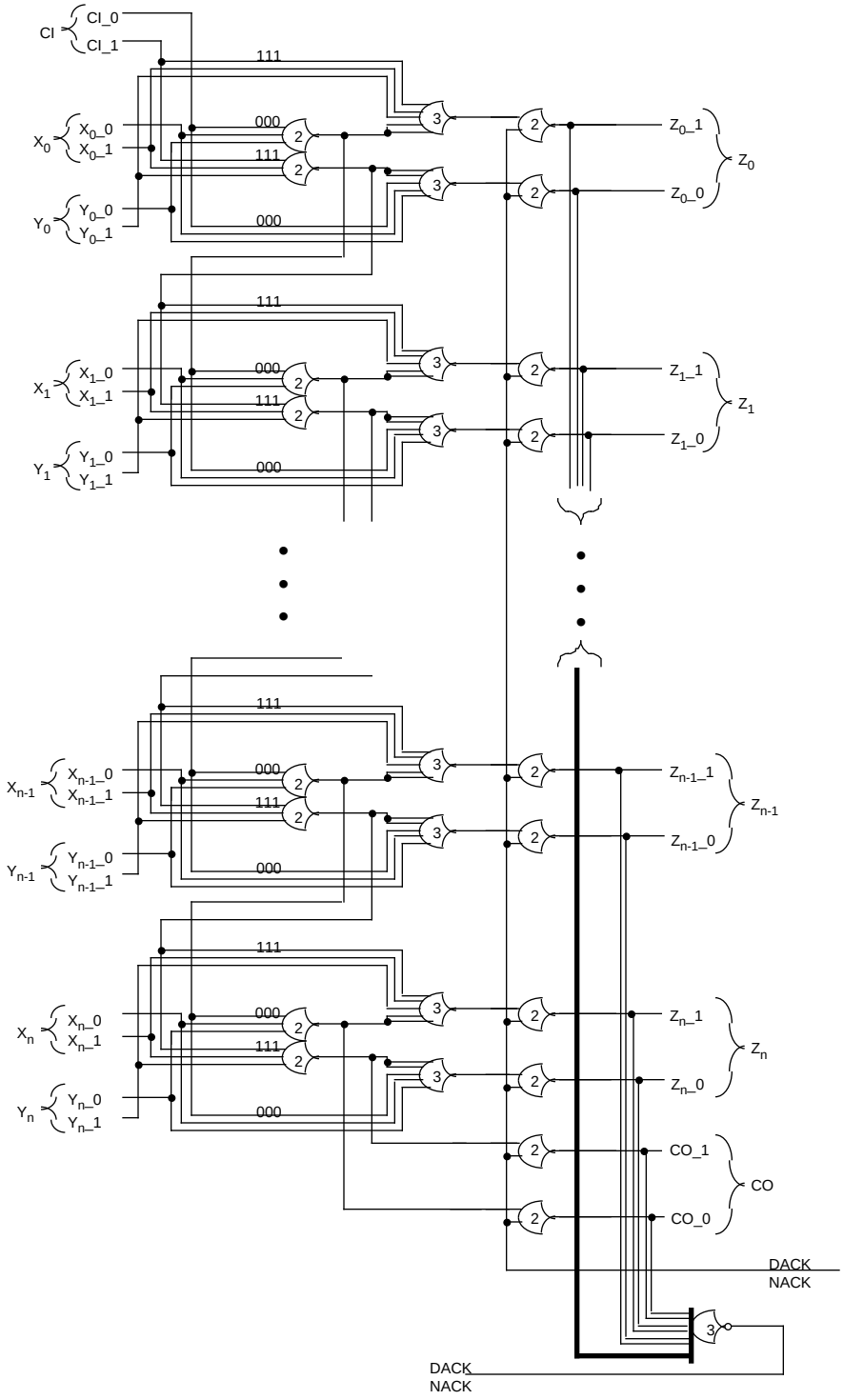

Figure 31. Ripple carry adder with output asynchronous register.

The point to appreciate is that the output register is just another NCL™ combinational circuit. There is no difference between the control expression of the circuit and the data processing aspect of the circuit and, indeed, they can be optimized in combination. Figure 32 shows such an optimization where the last rank of gates in the circuit are performing both the last stage of data processing and registration.

Looking back to the circuit in Figure 30 we see that the last rank of full adder gates are also hysteresis gates that maintain their state between wave fronts just like the asynchronous register gates. Since the wave front state is maintained by the adder gates there is really no need for a separate rank of gates just to maintain state (i.e. a separate asynchronous register). We can connect the acknowledge line directly to the last rank of processing gates and integrate the asynchronous registration function directly into the combinational circuit as shown in Figure 32.

With both input meanings and the acknowledge control signal going into the last rank of gates their threshold must now be 4.

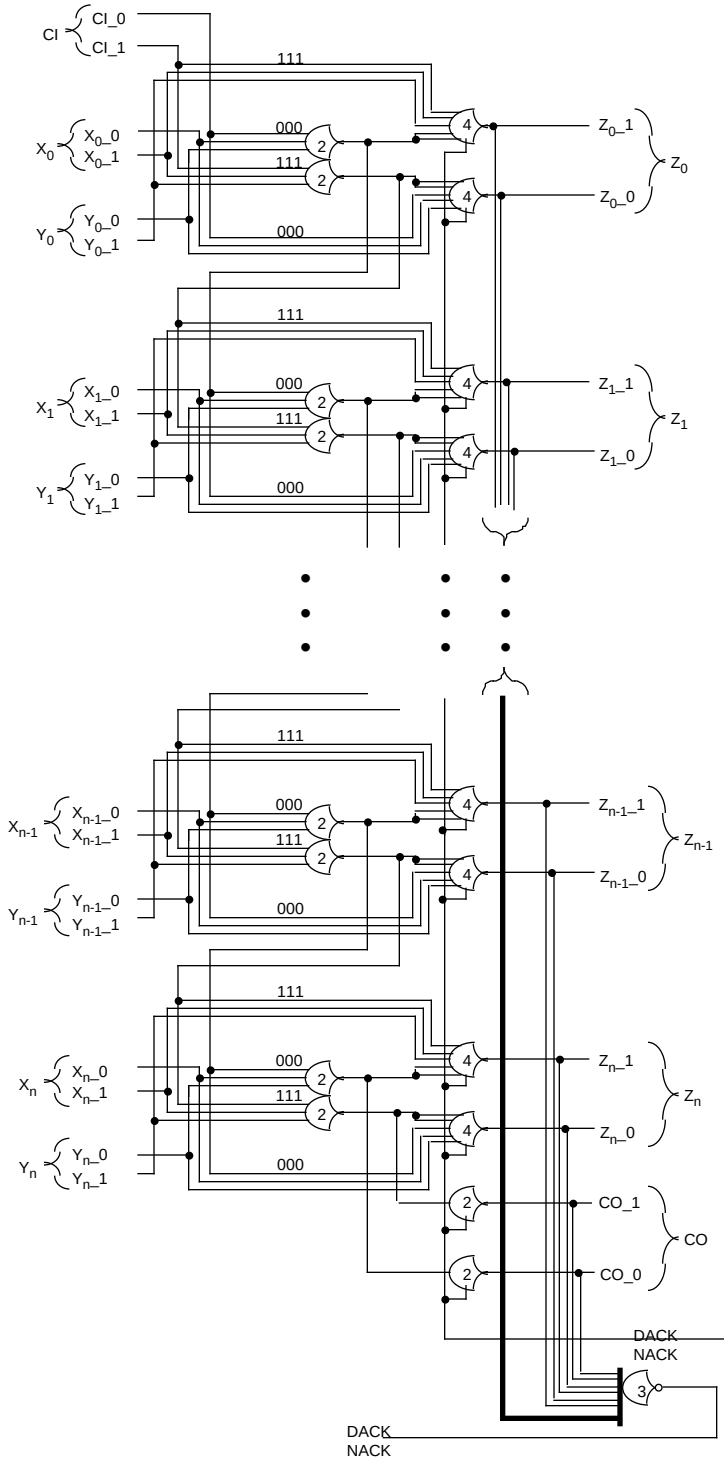

Figure 32. Ripple carry adder with embedded asynchronous register.

NULL Convention Logic™ expresses data processing and control identically and indeed the expressional distinction between the two domains disappears. NULL Convention Logic™ is a uniform, consistent and general language for the expression of asynchronous circuits and systems. NCL™ delivers on the promises of asynchronous design where the traditional Boolean logic based approaches have failed to deliver.

6. Summary

NULL Convention Logic™ conveniently and straightforwardly delivers all of the traditionally expected benefits of asynchronous circuit design.

EASE OF DESIGN

NULL Convention Logic™ circuits are purely symbolically determined in that circuit behavior depends solely on the interconnections among the gates. Delay insensitivity is inherent in the logic itself. It does not depend on subtle and expensive circuit constructs. Consequently an NCL™ circuit is a complete and autonomous expression in and of itself. It signals its own completion of processing and also signals its readiness to accept new input data, without any appeal to an external expression or authority such as a clock, delay line or controller. NCL™ circuits can be fully expressed in high level languages without regard for timing issues and compiled to silicon in the same sense that high level programs are compiled to machine instructions.

LOWER DESIGN COST AND RISK

The clock is eliminated with all its attendant design complexities and risks including clock skew. The system can be designed in parts and then directly composed. There are no global coordination issues as with synchronous systems.

LOWER POWER CONSUMPTION

Only portions of the system that are doing useful work consume power. Integrating logic function and registration in a single gate reduces the power required for system control. There is no spurious switching of transistors. The NULL state is an inherent and automatic power idle state. The clock driving power is eliminated. NCL™ systems operate entirely in terms of synchronized wave fronts of monotonic level transitions. There are no pulses or edge triggering involved in the circuit behavior. The asynchronous nature of NULL Convention Logic™ circuits distributes the demand for power.

CONVENIENT TECHNOLOGY MIGRATION

Because the logic is inherently delay insensitive it is insensitive to the behavior properties of the physical implementation. NULL Convention Logic™ circuits are insensitive to changes in implementation technology, to changes in scale, and to propagation delay changes due to aging or manufacturing variations eliminating portability and evolvability issues.

AUTOMATIC ADAPTATION TO PHYSICAL PROPERTIES

The delays of the various circuit elements change differentially with the change in physical parameters such as voltage, temperature, age, manufacturing variations and different implementation environments. Since the circuits are delay insensitive, they will continue to operate correctly over a large range of variation of these physical parameters.

RELIABILITY

All failure modes due to timing problems (race, hazards, skew, etc.) are eliminated. NCL™ provides advantages for design cost and risk, reliability of circuit performance and evolvability beyond the barriers of complexity and feature size facing clocked Boolean logic.

TESTING

Testing complexity is reduced in that stuck-at-1 faults halt the circuit. Only stuck-at-0 faults need to be exercised with applied patterns. Design time and risk as well as circuit testing requirements are expected to be decreased because of the elimination of the complexity of the clock with its critical timing issues.

SPEED OF OPERATION

NULL Convention Logic™ circuits are speed competitive even though they require two propagation cycles per unit of processing. They will operate at the full rate the logic and material allow and when appropriate will take advantage of average case propagation behavior. There are no margins added onto worst case propagation delays as with clocked circuits. Integration of the registration in logic gates allows more finely grained pipelining and consequently higher throughput rates than convention clocked techniques.

Two value NULL Convention Logic™ preserves all the advantages of traditional Boolean logic (two values, simple gates, straightforward synthesis) while, additionally, providing self determined, locally autonomous, self synchronizing, delay insensitive, and fault detecting behavior. Furthermore, NCL™ is compatible with the existing fabrication and design infrastructure and with existing clocked systems allowing convenient low cost market entry.

7. References

1. C. L. Seitz, “System Timing.” in Introduction to VLSI Systems, ed. by Carver Mead and Lynn Conway (Reading, Mass., Addison-Wesley, 1980), pp. 242-262.

2. Stephen H. Unger, Asynchronous Sequential Switching Circuits (New York, Wiley-Interscience, 1969)

3. Ilana David, Ran Ginosar and Michael Yoeli, "An Efficient Implementation of Boolean Functions as Self-Timed Circuits", IEEE Transactions on Computers , Vol. 41, No. 1, January 1992, pp. 2-10.

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

5. J. A. Brzozowski and C-J. H. Seger, "Advances in Asynchronous Circuit Theory Part I: Gate and Unbounded Inertial Delay Models", Bulletin of the European Association for Computer Science , Vol. 42, October 1990, pp. 198-248.

6. J. A. Brzozowski and C-J. H. Seger, "Advances in Asynchronous Circuit Theory Part II: Unbounded Inertial Delay Models, MOS Circuits, Design Techniques", Bulletin of the European Association for Computer Science , Vol. ??, Month Year, pp. 198-263.

7. Karl M. Fant and Scott A. Brandt, NULL Convention Logic™ System, US patent 5,305,463 April 19,1994.

8. H. Hulgaard and P. H Christensen, “Automated synthesis of Delay insensitive Circuits,” M.Sc. thesis (IDE 511), Dept of Computer Science, Tech. Univ. of Denmark, Lyngby, 1990.