Delay-Insensitive Gate-Level Pipelining

S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson

Keywords: Asynchronous logic design, self-timed circuits, dual-rail encoding, pipelining, NULL Convention Logic (NCL).

Abstract

Gate-Level Pipelining (GLP) techniques are developed to design throughput-optimal delay-insensitive digital systems using NULL Convention Logic (NCL). Pipelined NCL systems consist of *Combinational, Registration*, and *Completion* circuits implemented using threshold gates equipped with hysteresis behavior. NCL *Combinational* circuits provide the desired processing behavior between *Asynchronous Registers* that regulate wavefront propagation. NCL *Completion* logic detects completed DATA or NULL output sets from each register stage. GLP techniques cascade registration and completion elements to systematically partition a combinational circuit and allow controlled overlapping of input wavefronts. Both full-word and bit-wise completion strategies are applied progressively to select the optimal size grouping of operand and output data bits. To illustrate the methodology, GLP is applied to a case study of a 4-bit by 4-bit unsigned multiplier, yielding a speedup of 2.25 over the non-pipelined version, while maintaining delay-insensitivity.

1.0 Introduction

Even though delay-insensitive design methodologies do not utilize clocked control signals, they are still amenable to significant throughput increases by the pipelining of wavefronts. The objective of this paper is to develop and illustrate a pipelining methodology for maximizing throughput of delay-insensitive systems at the gate level. The delay-insensitive methodology used is NULL Convention Logic (NCL) [1].

1.1 Background

Pipelining facilitates temporal parallelism by partitioning a process into stages such that each stage operates simultaneously on different wavefronts of input operands. If a process that requires *N* time units can be partitioned into *S* identical stages then a steady-state throughput not to exceed S/N results per time unit may be realized. In practice numerous constraints, such as registration overhead between computational stages, limit the actual speedup achievable by pipelining. For instance, throughput limitations may be encountered as clocked Boolean circuits are partitioned to increasingly finer granularities. In particular, the clock period used to advance data between stages becomes increasingly dominated by the required design margins, including accommodations for clock skew. Clearly, asynchronous design methodologies need not provide design margins to accommodate clock skew. Nonetheless, they do possess their own constraints governing speedup by pipelining and can benefit substantially from optimized pipeline design strategies.

One approach to pipelining asynchronous circuits was described in Ivan Sutherland's work on *micropipelines* [2]. This method employs *two-phase handshaking* supporting transmission of *bundled data*. Figure 1 shows a two-phase handshaking protocol. Two control wires, labeled *request* and *acknowledge*, are used to support an arbitrary number of data wires. In two-phase handshaking, both the rising and falling edges of the request and acknowledge signals are indicative of circuit behavior. A cycle begins with the sender setting the data lines and generating a request event by toggling the request line. When the request is received, the data is latched and the receiver generates an acknowledge event by toggling the acknowledge line. The cycle terminates when the sender receives the acknowledge signal, at which time the data lines may be set for the next cycle. The use of bundled data refers to the fact that the data lines and request signal are treated as a bundle. Data bundling implies that the data transmission delay cannot exceed the delay to transmit the request. Otherwise, the request event might reach the receiver prior to valid data, causing invalid data to be latched. Subsequent work on micropipelines [3, 4, 5] suggests that performance may be increased by using four-phase handshaking protocols. Four-phase handshaking also requires two control wires, *request* and *acknowledge*, along with an arbitrary number of data wires. But, in four-phase handshaking only one edge, either the rising or falling edge of the request and acknowledge signals, is active. The four-phase handshaking protocol is shown in Figure 2, using the rising edge as active. A cycle begins with the sender placing data on the bus and generating a request event by asserting the request line. When the request is received, the data is latched and the receiver generates an acknowledge event by asserting the acknowledge line. When the sender receives the acknowledge signal, the request signal is de-asserted and the data lines may be set for the next cycle. The cycle concludes with the acknowledge line being de-asserted, as precipitated by the de-assertion of the request line. Micropipelining techniques such as these are evident in several processors that have been designed and implemented using bundled data methods [6, 7].

Another approach to pipelining asynchronous circuits is through the use of *wave pipelining*. Hauck and Huss [8] describe a technique that allows multiple data wavefronts to simultaneously propagate between two asynchronous registers by partitioning each combinational logic block with dynamic latches, controlled only by the request line. Synchronous wave pipelining and asynchronous micropipelining methods can be combined using these techniques. However, a potential limitation of eliminating the acknowledge signal is that delay-insensitive behavior may be compromised, thus making the protocol inelastic. Further work by Park and Chung [9] presents a modification to this approach in which both the number of latches and the number of delay elements can be reduced, resulting in higher throughput.

A third asynchronous pipelining approach uses delay-insensitive *multi-ring structures* [10]. This method employs a four-phase handshaking protocol using dual-rail signals for data representation and Delay-Insensitive Minterm Synthesis (DIMS) [11] for each functional block. It also presents a formal method for analyzing the performance of these multi-ring structures, based on signal transition graphs. Nonetheless, formal methods to design throughput-optimal multi-ring structures are not directly feasible due to underlying difficulties in partitioning of DIMS expressions.

In [12] Kim and Beerel present an optimal branch and bound algorithm to partition asynchronous circuits composed of precharge-logic blocks [13, 14] designed at the transistor level. The algorithm uses a labeled directed graph to represent the model being pipelined. However, this method is not directly amenable to pipelining NCL circuits due to the differences in the fundamental components. In Section 2.5, delay-insensitive design strategies based on NCL will be shown to be directly amenable to partitioning and will be compared to the alternative approaches described above, in Section 2.6.

1.2 Paper Outline

This paper is organized into five sections. An overview of NCL is given in Section 2. In Section 3, the GLP methodology is developed. This method is demonstrated in Section 4 by applying GLP to design an optimal 4-bit by 4-bit unsigned multiplier whose throughput is increased by 125% over the non-pipelined version. Section 5 provides conclusions and outlines directions for future work.

2.0 Overview of NCL

NULL Convention Logic (NCL) provides an asynchronous design methodology employing dual-rail signals, quad-rail signals, or other Mutually Exclusive Assertion Groups (MEAGs) to incorporate data and control information into one mixed path. In NCL, the control is inherently present with each datum, so there is no need for worse-case delay analysis and control path delay matching. NCL follows the so-called "weak conditions" of Seitz's delay-insensitive signaling scheme [15]. As with other delay-insensitive logic methods discussed herein, the NCL paradigm assumes that forks in wires are isochronic [16, 17]. The origins of various aspects of the paradigm, including the NULL (or spacer) logic state from which NCL derives its name, can be traced back to Muller's work on speed-independent circuits in the 1950s and 1960s [18]. Earlier work by Seitz presents an extensive discussion of self-timed logic, illustrating its advantages over traditional clocked logic, and includes one approach to designing such circuits [15]. Some other methods of designing delay-insensitive circuits are detailed in [19, 20, and 21]. These techniques concentrate on developing circuits from a standardized set of gates, while other techniques [22, 23] emphasize formal logic methods that directly yield designs at the transistor-level.

Various design aspects of NCL were patented by Karl Fant and Scott Brandt in April of 1994 [24]. Acknowledging that clocked circuits unnecessarily restricted execution flow, consumed power proportional to the operating frequency, occupied significant device area for the clock tree, and greatly complicated the design process, they sought a clockless design approach. But eliminating clocks as in traditional asynchronous design presented race conditions and made timing optimizations like pipelining difficult. By eliminating clocks but retaining control information in the datapath, NCL aims at designing VLSI devices with greater ease, with a reduced power budget, lower electromagnetic interference effects, and reduced noise margins.

2.1 Delay-Insensitivity

NCL uses symbolic completeness of expression [1] to achieve delay-insensitive behavior. A symbolically complete expression is defined as an expression that only depends on the relationships of the symbols present in the expression without a reference to the time of evaluation. Traditional Boolean logic is not symbolically complete; the output of a Boolean gate is only valid when referenced with time. For example, assume it takes 1 ns for output Z of an AND gate to become valid once its inputs X and Y have arrived. As shown in Figure 3, suppose $X = 1$, $Y = 0$, and $Z = 0$, initially. If Y changes to 1, Z will change to 1 after 1 ns; so Z is not valid from the time Y changes until 1 ns later. Therefore output Z not only depends on the inputs X and Y, but time must also be referenced in order to determine the validity of Z. This can be critical when Z is used as an input to another circuit.

In particular, dual-rail signals, quad-rail signals, or other *Mutually Exclusive Assertion Groups* (MEAGs) can be used to incorporate data and control information into one mixed signal path to eliminate time reference [25]. A dual-rail signal, D, consists of two wires, D^0 and D^1, which may assume any value from the set {DATA0, DATA1, NULL}. The DATA0 state ($D^0 = 1$, $D^1 = 0$) corresponds to a Boolean logic 0, the DATA1 state ($D^0 = 0$, $D^1 = 1$) corresponds to a Boolean logic 1, and the NULL state ($D^0 = 0$, $D^1 = 0$) corresponds to the empty set meaning that the value of D is not yet available. The two rails are mutually exclusive, so that both rails can never be asserted simultaneously; this state is defined as an illegal state. A quad-rail signal, Q, consists of four wires, Q^0, Q^1, Q^2, and Q^3, which may assume any value from the set {DATA0, DATA1, DATA2, DATA3, NULL}. The DATA0 state ($Q^0 = 1$, $Q^1 = 0$, $Q^2 = 0$,

$Q^3 = 0$) corresponds to two Boolean logic signals, *X* and *Y*, where X = 0 and Y = 0. The DATA1 state ($Q^0 = 0$, $Q^1 = 1$, $Q^2 = 0$, $Q^3 = 0$) corresponds to X = 0 and Y = 1. The DATA2 state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 1$, $Q^3 = 0$) corresponds to X = 1 and Y = 0. The DATA3 state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 1$) corresponds to X = 1 and Y = 1, and the NULL state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 0$) corresponds to the empty set meaning that the result is not yet available. The four rails of a quad-rail NCL signal are mutually exclusive, so no two rails can ever be asserted simultaneously; these states are defined as illegal states. Both dual-rail and quad-rail signals are space optimal delay-insensitive codes, requiring two wires per bit. Other higher order MEAGs are not wire count optimal, however they can be more power efficient due to the decreased number of transitions per cycle.

Consider the behavior of a symbolically complete AND function using NCL as shown in Figure 4. Assume it takes 1 ns for output *Z* of a NCL AND function to become valid once its inputs *X* and *Y* have arrived. Also, initially suppose *X* is DATA1, *Y* is DATA0, and *Z* is DATA0. Before the next set of inputs can be applied, all inputs must first transition to NULL, which causes the output to transition to NULL, 1 ns later. Once the output has transitioned to NULL, the next input set can be applied. If the next input set consists of *X* = DATA1 and *Y* = DATA1, *Z* will become DATA1 after 1 ns, signaled by *Z* transitioning from NULL to DATA. Output *Z* will remain DATA1 until both inputs, *X* and *Y*, transition to NULL, due to the hysteresis behavior inherent in each threshold gate. Time is never referenced to determine the validity of *Z*. The 1 ns delay is an arbitrary gate transition delay and does not affect the validity of *Z*.

2.2 Logic Gates

NCL gates are a special case of the logical operators or gates available in digital VLSI circuit design [26]. Such an operator consists of a set condition and a reset condition that the environment must ensure are not both satisfied at the same time. If neither condition is satisfied then the operator maintains its current state. NCL uses *threshold gates* with *hysteresis* [27] for its composable logic elements. One type of threshold gate is the *THmn gate*, where $1 \leq m \leq n$, as depicted in Figure 5. A THmn gate corresponds to an operator with at least m signals asserted as its set condition and all signals de-asserted as its reset condition. THmn gates have *n* inputs. At least *m* of the *n* inputs must be asserted before the output will become asserted. Because threshold gates are designed with hysteresis, all asserted inputs must be de-asserted before the output will be de-asserted. Hysteresis is used to provide a means for monotonic transitions and a complete transition of multi-rail inputs back to a NULL state before asserting the output associated with the next wavefront of input data. In a THmn gate, each of the *n* inputs is connected to the rounded portion of the gate. The output emanates from the pointed end of the gate. The gate's threshold value, *m*, is written inside of the gate. [27] details various design implementations (static, semi-static, and dynamic) of THmn gates.

By employing threshold gates for each logic rail, NCL is able to determine the output status without referencing time. Inputs are partitioned into two separate wavefronts, the NULL wavefront and the DATA wavefront. The NULL wavefront consists of all inputs to a circuit being NULL, while the DATA wavefront refers to all inputs being DATA, some combination of DATA0 and DATA1. Initially all circuit elements are reset to the NULL state. First, a DATA wavefront is presented to the circuit. Once all of the outputs of the circuit transition to DATA, the NULL wavefront is presented to the circuit. Once all of the outputs of the circuit transition to NULL, the next DATA wavefront is presented to the circuit. This DATA/NULL cycle continues repeatedly. As soon as all outputs of the circuit are DATA, the circuit's result is valid. The NULL wavefront then transitions all of these DATA outputs back to NULL. When they transition back to DATA again, the next output is available. This period is referred to as the DATA-to-DATA cycle time, denoted as T_{DD} and has an analogous role to the clock period in a synchronous system.

2.3 Completeness of Input

The *input-completeness* criterion [1], which NCL circuits must maintain in order to be delay-insensitive, requires that:

1. the outputs of a circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and
2. the outputs of a circuit may not transition from DATA to NULL until all inputs have transitioned from DATA to NULL.

In circuits with multiple outputs, it is acceptable for some of the outputs to transition without having a complete input set present, as long as all outputs cannot transition before all inputs arrive. This signaling scheme is equivalent to the "weak conditions" of delay-insensitive signaling defined by Seitz [15]. Consider the incomplete NCL AND function shown in Figure 6. The output can change from NULL to DATA0 without both inputs first transitioning to DATA. For instance, if A = DATA0 and B = NULL then C = DATA0, which breaks the completeness of input criterion. Figure 7 shows a complete NCL AND function since the output cannot transition until both inputs have transitioned.

2.4 Observability

There is one more condition that must be met in order for NCL to retain delay-insensitivity. No *orphans* may propagate through a gate. An orphan is defined as a wire that transitions during the current DATA wavefront, but is not used in the determination of the output. Orphans are caused by wire forks and can be neglected through the isochronic fork assumption, as long as they are not allowed to cross a gate boundary. This *observability* condition ensures that every gate transition is observable at the output. Consider an incorrect version of an XOR function shown in Figure 8, where an orphan is allowed to pass through the TH12 gate. For instance, when X = DATA0 and Y = DATA0, the TH12 gate is asserted, but does not take part in the determination of the output, Z = DATA0. This orphan path is shown in boldface in Figure 8. A correct, fully observable version of the XOR function is given in Figure 9, where no orphans propagate through any gate. An orphan checker tool, as a Synopsys shell, is run on each design to ensure observability.

2.5 Pipelining in NCL

As shown in Figure 10, pipelined NCL systems consist of cascaded arrangements of three main functional blocks, *Registration*, *Completion*, and *Combinational* circuits [1]. The NCL *Register* controls the DATA/NULL wavefronts. NCL *Completion* detects complete DATA and NULL sets, where all outputs are DATA or all outputs or NULL, respectively, at the output of every register stage. NCL *Combinational* circuits provide the desired input/output processing behavior, as detailed in [28].

The design of the NCL registration stage is discussed first. The single-bit dual-rail NCL register [1], shown in Figure 11, controls the DATA/NULL wavefronts through its *request in* and *request out* lines, K_i and K_o, respectively. When either K_i or K_o is asserted it is requesting DATA, denoted *rfd*; and when either K_i or K_o is de-asserted it is requesting NULL, denoted *rfn*. Assume the register output is initially NULL and K_i is initially *rfn*. Due to the NULL output, K_o will initially be *rfd*. A DATA value at the input will not be able to pass to the output of the register until K_i becomes *rfd*. Once K_i is *rfd*, the DATA value at the input passes through the register to the output and causes K_o to become *rfn*. Likewise, a NULL value at the input will not be able to pass to the output of the register until K_i becomes *rfn*, due to the hysteresis functionality of the TH22 gates. Once K_i is *rfn*, the NULL value at the input passes through the register to the output and causes K_o to once again become *rfd*. The actual register includes reset circuitry not shown in Figure 11, to reset the output to DATA0, DATA1, or NULL. Single-bit NCL registers can then be connected as shown in Figure 12 to form a pipeline. This design is representative of a First-In First-Out (FIFO) buffer.

The previous example only considered single-bit NCL registers. Now consider an *N*-bit register stage, comprised of *N* single-bit NCL registers. Clearly, there will now be *N* completion signals required, one for each bit. The NCL *Completion* component, shown in Figure 13, uses these N K_o lines to detect complete DATA and NULL sets at the output of every register stage and request the next NULL and DATA sets, respectively. The single-bit output of the completion component is connected to all K_i lines of the previous register stage. Since the maximum input threshold gate currently supported is the TH44 gate, the number of logic levels in the completion component for an *N*-bit register is given by $\lceil Log_4 N \rceil$.

All NCL systems have at least two register stages, one at both the input and output. These two register stages interact through their K_i and K_o lines to prevent DATA set_i from overwriting DATA $\text{set}_\text{i-1}$ by ensuring that the two DATA sets are always separated by a NULL set. Even though these systems are self-timed, it is possible to take advantage of pipelining techniques when interconnecting NCL registration, completion, and combinational circuits.

2.6 Relation of NCL to Previous Work

For Sutherland's *micropipelines* using either two-phase or four-phase handshaking, the determination of the maximum throughput design for a given combinational circuit is straightforward. Since micropipelines assume bundled data and therefore employ single-rail signals, there is no completeness of input criterion that must be met when partitioning a circuit, therefore further partitioning cannot invalidate a design. Furthermore, delay is added in the control path such that completion detection is unnecessary, therefore further partitioning cannot decrease throughput. Thus, the design that will yield the maximum throughput is the one containing only one gate delay per stage. Since micropipelines necessitate the addition of delay in the control path, they exhibit worse-case performance verses the average-case performance of NCL systems and are layout and process dependent unlike NCL systems. Micropipelines also assume bundled data such that synchronicity is required while NCL systems require no synchronization so that inputs may arrive at any time and in any order, therefore NCL systems are potentially more independent than micropipelines.

Since the maximum throughput rate for *asynchronous wave pipelines* is determined by the difference between the longest and shortest path through the combinational logic, there is even more timing analysis required than for micropipelines. In asynchronous wave pipelines throughput will be maximized by designing the shortest and longest path to be nearly equal, therefore extensive timing analysis is required. Asynchronous wave pipelines are therefore very susceptible to process dependencies and environmental variations, unlike NCL. These fundamental differences between NCL and both micropipelines and asynchronous wave pipelines place NCL in a different class than either and would make direct comparisons difficult.

NCL circuits are in the same class as other delay-insensitive approaches [15, 19, 20, and 21], which were compared to NCL in [28]. The functionality of NCL circuits is the same as those designed using the approaches presented in [15, 19, 20, and 21]. Thus, the NCL combinational circuit, as part of the NCL gate-level pipelining framework, could be replaced with an equivalent circuit designed using [15, 19, 20, or 21], and the resulting single-stage system would function correctly. This is exactly what delay-insensitive multi-ring structures are. Their framework is equivalent to that of NCL, except for the combinational circuits, which use the approach described in [11]. But, since all of the basic gates used in the other delay-insensitive approaches, including delay-insensitive multi-ring structures, do not include hysteresis, their combinational designs cannot be partitioned, as can NCL combinational circuits. Thus, a given combinational circuit designed using [15, 19, 20, or 21] can either be used as a non-pipelined design, or if increased throughput is desired, each stage of the pipeline must be separately redesigned. Therefore a method that iteratively divides a combinational circuit of a delay-insensitive multi-ring structure to increase throughput cannot do so easily, as does the method presented herein for NCL; since after each iteration all combinational blocks that were divided would have to be redesigned to include input-completeness necessary for delay-insensitivity.

3.0 Methodology Definition

In [28] it was shown how to design an optimal NCL combinational circuit. So, starting with an N-level NCL combinational logic design, the design process for optimizing throughput begins, as depicted in Figure 14. Other criteria such as maximum latency and maximum area may also be considered during throughput optimization. Several alternate designs are generated which are then assessed against the optimization criteria, allowing the preferred design to be selected for implementation.

It is assumed that if a maximum latency bound is specified then it is at least one stage, and that if a maximum area bound is specified then it is at least as large as the non-pipelined design, otherwise the non-pipelined design will be output. If no maximum latency or maximum area requirements are specified, then both are assumed to be infinity such that they are not considered in determining the optimal design. If more than one design has the same throughput, the one with the least latency will be chosen. If multiple designs have the same throughput and latency, the one with the least area will be chosen.

The original combinational circuit with no pipelining will always be input complete since [28] only yields input complete designs. Thus, starting with the combinational logic design and adding registration along with corresponding completion logic at the input and output will yield an initial 1-stage design. Partitioning this initial design, first into 2 stages, then further into as many as *N* stages may or may not produce better designs. First, completeness of input must be ensured at the output of each stage, as discussed in Section 2.3, otherwise the design will not be delay-insensitive and therefore invalid. After input completeness is ensured, the throughput for the current design must be calculated and compared to the throughput of the best design. If the current design's throughput is greater than that of the best design, it is designated as the best design, otherwise bit-wise completion is applied to the current design and the throughput is reevaluated. If the throughput of the current design using bit-wise completion is still not greater than that of the best design, the best design does not change since the current design doesn't increase throughput and has longer latency, otherwise the current design using bit-wise completion becomes the best design. As mentioned in Section 2.5 the completion delay is proportional to $\lceil \mathrm{Log}_4 \mathrm{N} \rceil$. Thus, if partitioning causes registers of significantly larger width to be required then the decrease in the combinational delay per stage will be offset by the increase in the completion delay such that the throughput of the system may not necessarily increase, as discussed in Section 3.1. If after traversing the loop of Figure 14 (`i=0`), which generates each subsequent pipelined design, or if the maximum latency or area requirements have been exceeded, then if the best design utilizes full-word completion, bit-wise completion is applied to this design to possibly further increase throughput. If throughput is not increased the design with the least area is chosen since both designs will have the same throughput and latency. This is because application of bit-wise completion won't decrease throughput, as explained in Section 3.2, and doesn't impact the number of stages. The output of this flowchart will be the optimal design (`best_design`) that produces the maximum throughput (`max_throughput`), and does not exceed the maximum latency or maximum area requirements, if any were given.

3.1 Throughput Derivation

Quarter-cycle timing is used to determine the worst-case achievable throughput of a pipelined NCL system. The name is derived from the fact that the analysis requires each NCL cycle to be broken into its four sub-cycles. The NCL cycle is comprised of the DATA and NULL propagation through the combinational circuitry, as well as the generation of the request for DATA and request for NULL from the completion circuitry. The four sub-cycles that are contained in the NCL cycle are shown in Figure 15. *D* denotes the interval when any DATA bits are propagating through the combinational circuit, *N* denotes the interval when any NULL bits are propagating through the combinational circuit, *RFD* is the request for DATA generation, and *RFN* is the request for NULL generation. Assuming $K_o = rfd$, the cycle starts with DATA propagation and the sequence of the four sub-cycles is as follows: D, RFN, N, and RFD. The propagation delays associated with this sequence are labeled as follows: TD, TRFN, TN, and TRFD, respectively. TD and TN are defined to be the delay experienced by the slowest bit through their respective sub-cycles. For this paper TD, TRFN, TN, and TRFD are calculated in terms of gate delays, making the predicted throughput an estimate since different gates do have slightly different delays. When this methodology is automated the actual delay of each gate will be used to calculate the predicted throughput.

The NCL cycle is bounded by the current registration stage, denoted as *i*, and the previous registration stage, denoted by *i-1*, as depicted in Figure 16. The calculation resulting in the maximum cycle time forms a lower bound on the throughput of the i^{th} and $i\text{-}1^{th}$ registration pair. This process of bounding the throughput for registration pairs is repeated for all adjacent registration pairs in a pipelined configuration. The maximum value calculated over all adjacent registration pairs determines a lower bound on steady-state throughput for the entire design.

3.1.1 Idealized Completion Circuitry

Consider the idealized case where TRFN and TRFD are assumed to be zero. The discrete timing chart in Table I identifies the interaction of stage_i and stage_{i-1} under these idealized conditions. For the initial state, the analysis begins with stage_i and stage_{i-1} both reset to NULL. At wavefront #1, DATA propagates through the combinational circuitry of stage_{i-1}, while stage_i remains idle. At wavefront #2, NULL propagates through the combinational circuitry of stage_{i-1}, while DATA propagates through the combinational circuitry of stage_i. At wavefront #3, DATA propagates through the combinational circuitry of stage_{i-1}, while NULL propagates through the combinational circuitry of stage_i. This pattern of NULL propagating through stage_{i-1}, while DATA propagates through stage_i, followed by DATA propagating through stage_{i-1}, while NULL propagates through stage_i, repeats continuously and forms the simplified NCL cycle, shown in boldface in Table I.

Using the above terminology, the worst-case DATA-to-DATA cycle time for stage_i using idealized completion is:

$$T_{DD_i}^{\text{idealized}} = \text{MAX}\,(TN_{i-1}, TD_i) + \text{MAX}\,(TD_{i-1}, TN_i) \qquad \text{(eq. 3.1).}$$

Interpreting Equation 3.1 as a set of exclusive events implies exactly one of the following relationships:

either $T_{DD_i}^{\text{idealized}} = TN_{i-1} + TD_{i-1}$ (eq. 3.2), or

$T_{DD_i}^{\text{idealized}} = TN_{i-1} + TN_i$ (eq. 3.3), or

$T_{DD_i}^{\text{idealized}} = TD_i + TD_{i-1}$ (eq. 3.4), or

$T_{DD_i}^{\text{idealized}} = TD_i + TN_i$ (eq. 3.5).

Notice that Equations 3.2 and 3.5 are equivalent except for their stage index. Under the proposed method of evaluating each stage pair in increasing order to determine the global maximum value, Equation 3.2 would therefore have been evaluated in the previous registration pair calculations, so it does not need to be reevaluated in the current registration pair calculations. This is true for every registration pair except the first pair, stage 1 and stage 2. For the first registration pair, Equation 3.2 does need to be considered since there is no previous registration pair that incorporates this calculation.

Equation 3.3 considers the case of adjacent NULL propagation delays. Equation 3.4 considers the case of adjacent DATA propagation delays. Equation 3.5 considers the case of NULL and DATA propagation delays for a single registration stage. The pseudocode listed in Algorithm 3.1 calculates the worst-case throughput for an idealized N-stage NCL pipeline.

```
max_cycle_time = $TD_1 + TN_1$
for (i = 2 to N) loop
   temp_cycle_time = MAX($TN_{i-1} + TN_i$, $TD_{i-1} + TD_i$, $TD_i + TN_i$)
   if (temp_cycle_time > max_cycle_time) then
      max_cycle_time = temp_cycle_time
   end if
end loop
worst_case_throughput = 1 / max_cycle_time
```

Algorithm 3.1: Calculation of worst-case throughput for an idealized N-stage NCL pipeline.

Evaluation of the above loop is followed by taking the reciprocal of the maximum adjacent stage pair delay to obtain a lower bound on the pipeline's throughput.

3.1.2 Non-Zero Delay Completion Circuitry

Now the general case will be examined, where TRFN and TRFD are not zero. The discrete timing chart in Table II shows the interaction of stage_i and stage_{i-1}. For the initial state, assume stage_i and stage_{i-1} are both reset to NULL, so both stages will initially be requesting DATA. At wavefront #1, DATA propagates through the combinational circuitry of stage_{i-1}, while stage_i remains idle. At wavefront #2, DATA propagates through the combinational circuitry of stage_i, while stage_{i-1} requests NULL. At wavefront #3, NULL propagates through the combinational circuitry of stage_{i-1}, while stage_i requests NULL. At wavefront #4, NULL propagates through the combinational circuitry of stage_i, while stage_{i-1} requests DATA. At wavefront #5, DATA propagates through the combinational circuitry of stage_{i-1}, while stage_i requests DATA. This pattern, from wavefront #2 to wavefront #5, repeats continuously and forms the generalized NCL cycle, shown in boldface in Table II.

The worst-case cycle time for the generalized case of stage_i is then given by:

$$\begin{aligned} T_{DD_i} = {} & \text{MAX}(TD_i, TRFN_{i-1}) + \text{MAX}(TN_{i-1}, TRFN_i) + \\ & \text{MAX}(TN_i, TRFD_{i-1}) + \text{MAX}(TD_{i-1}, TRFD_i) \qquad \text{(eq. 3.6).} \end{aligned}$$

Interpreting Equation 3.6 as a set of exclusive events implies exactly one of the following relationships:

either $T_{DD_i} = TD_i + TN_{i-1} + TN_i + TD_{i-1}$ (eq. 3.7), or

$T_{DD_i} = TD_i + TN_{i-1} + TN_i + TRFD_i$ (eq. 3.8), or

$T_{DD_i} = TD_i + TN_{i-1} + TRFD_{i-1} + TD_{i-1}$ (eq. 3.9), or

$T_{DD_i} = TD_i + TN_{i-1} + TRFD_{i-1} + TRFD_i$ (eq. 3.10), or

$T_{DD_i} = TD_i + TRFN_i + TN_i + TD_{i-1}$ (eq. 3.11), or

$T_{DD_i} = TD_i + TRFN_i + TN_i + TRFD_i$ (eq. 3.12), or

$T_{DD_i} = TD_i + TRFN_i + TRFD_{i-1} + TD_{i-1}$ (eq. 3.13), or

$T_{DD_i} = TD_i + TRFN_i + TRFD_{i-1} + TRFD_i$ (eq. 3.14), or

$T_{DD_i} = TRFN_{i-1} + TN_{i-1} + TN_i + TD_{i-1}$ (eq. 3.15), or

$T_{DD_i} = TRFN_{i-1} + TN_{i-1} + TN_i + TRFD_i$ (eq. 3.16), or

$T_{DD_i} = TRFN_{i-1} + TN_{i-1} + TRFD_{i-1} + TD_{i-1}$ (eq. 3.17), or

$T_{DD_i} = TRFN_{i-1} + TN_{i-1} + TRFD_{i-1} + TRFD_i$ (eq. 3.18), or

$T_{DD_i} = TRFN_{i-1} + TRFN_i + TN_i + TD_{i-1}$ (eq. 3.19), or

$T_{DD_i} = TRFN_{i-1} + TRFN_i + TN_i + TRFD_i$ (eq. 3.20), or

$T_{DD_i} = TRFN_{i-1} + TRFN_i + TRFD_{i-1} + TD_{i-1}$ (eq. 3.21), or

$T_{DD_i} = TRFN_{i-1} + TRFN_i + TRFD_{i-1} + TRFD_i$ (eq. 3.22).

Observe that Equations 3.17 and 3.12 are equivalent except for their stage index, as in the simplified case. Thus, Equation 3.17 would have been evaluated in the previous registration pair calculations, so it does not need to be reevaluated in the current registration pair calculations, except for the first pair, stage 1 and stage 2. Equations 3.7 through 3.11, 3.14, 3.15, and 3.18 through 3.22, inclusive, can also be omitted based on the fact that they contain terms with overlapping time intervals. For example, consider Equation 3.11 containing TN_i, then from Equation 3.6, $TN_i > TRFD_{i-1}$, which means that RFD_{i-1} completes before N_i. Since D_{i-1} can begin as soon as RFD_{i-1} completes and RFD_{i-1} completes before N_i, then the intervals labeled D_{i-1} and N_i must at least partially overlap. Thus, Equation 3.11 can be disregarded since it does not take into account overlap. To remove the overlap, TN_i could be replaced with $TRFD_{i-1}$, which would yield the existing equation, 3.13. Through a similar analysis, three other overlapping terms can be found. Therefore any equation containing one or more of these overlapping pairs: TN_i and TD_{i-1}, TD_i and TN_{i-1}, $TRFN_i$ and $TRFN_{i-1}$, or $TRFD_i$ and $TRFD_{i-1}$ must also be invalid, leaving only three valid equations, 3.12, 3.13, and 3.16.

In particular, Equation 3.16 considers the case of adjacent NULL propagation delays, including the request times. Equation 3.13 considers the case of adjacent DATA propagation delays, including the request times. Equation 3.12 considers the case of NULL and DATA propagation delays for a single registration stage, including the request times. Based on this analysis, the pseudocode listed in Algorithm 3.2 can be used to calculate the worst-case throughput for a generalized N-stage NCL pipeline.

```
max_cycle_time = $TRFD_1 + TD_1 + TRFN_1 + TN_1$
for (i = 2 to N) loop
     temp_cycle_time = MAX($TRFD_i + TD_i + TRFN_i + TN_i$,
                           $TRFD_{i-1} + TD_{i-1} + TD_i + TRFN_i$,
                           $TRFN_{i-1} + TN_{i-1} + TN_i + TRFD_i$)
     if (temp_cycle_time > max_cycle_time) then
          max_cycle_time = temp_cycle_time
     end if
end loop
worst_case_throughput = 1 / max_cycle_time
```

Algorithm 3.2: Calculation of worst-case throughput for a generalized N-stage NCL pipeline.

Evaluation of the above loop is followed by taking the reciprocal of the maximum delay to obtain a lower bound on the pipeline's throughput.

3.2 Bit-Wise Completion

In addition to minimizing stage delay, throughput may be further increased using *bit-wise completion*. Until now only full-word completion has been utilized, where the completion signal for each bit in $register_i$ is conjoined by the completion component, whose single-bit output is connected to all K_i lines of $register_{i-1}$. On the other hand, bit-wise completion only sends the completion signal from bit *b* in register$_i$ back to the bits in register$_{i-1}$ that took part in the calculation of bit *b*. This method may therefore require fewer logic levels than that of full-word completion, thus increasing throughput. Bit-wise completion will never reduce throughput, since in the worse case all bits of register$_{i-1}$ are used to calculate each bit of register$_i$, such that the completion logic and therefore throughput does not change by selecting bit-wise completion rather than full-word completion. Bit-wise completion may or may not require more logic gates and therefore transistors than full-word completion, thus bit-wise completion will be used if it increases throughput, or if throughput is the same as for full-word completion but area is reduced.

Figure 17 shows full-word completion for a combinational stage of six 2-input AND functions, using all combinations of the 4-bit input *X*. Figure 18 shows bit-wise completion for the same six AND functions. There is only one level of logic in the completion components for the bit-wise completion approach verses two levels of logic in the completion component for the full-word completion approach. Also notice that four gates are required for bit-wise completion verses three gates for full-word completion, a difference of 8 additional transistors. To maximize throughput in this case, bit-wise completion would be selected in spite of its larger size since it reduces the completion logic path from two gate delays down to only one gate delay, which translates to an increase in throughput by the procedure given in Section 3.1.2.

4.0 Application to Unsigned Multiplier

A number of designs based on the 4-bit by 4-bit multiplier shown in Figure 19 have been evaluated as a case study to assess the impact of GLP methods on throughput. The specifications for this multiplier were simply to perform an unsigned multiply of the two 4-bit input vectors, *X* and *Y*, and then output their 8-bit product, *S*. As with all NCL circuits, a full NCL interface with request and acknowledge signals labeled K_i and K_o, respectively, is included for requesting and acknowledging complete DATA and NULL wavefronts.

The non-pipelined version of the 4x4 multiplier is shown in Figure 20. It consists of incomplete AND functions, denoted as *I* and depicted in Figure 6, as well as complete AND functions, denoted as *C* and shown in Figure 7. The multiplier also utilizes half-adders, as shown in Figure 21 and denoted *HA*, as well as full-adders, as shown in Figure 22 and denoted *FA*. The last components of the multiplier include *GEN_S7*, as shown in Figure 23, and the completion components, denoted *COMP*. Remember that the number of gate delays in the completion logic for an N-bit register is $\lceil \text{Log}_4 \text{ N} \rceil$, as discussed in Section 2.5.

4.1 Pipelined Multipliers with Full-Word Completion

The throughput for the non-pipelined design is calculated using the pseudocode in Section 3.1.2, and is determined to be (24 gate delays)$^{-1}$. Here, $\text{TRFD}_1 = \text{TRFN}_1 = \lceil \text{Log}_4 \text{ } 8 \rceil = 2$ gate delays and $\text{TN}_1 = \text{TD}_1 = 10$ gate delays as given by the I, FA, FA, HA, FA, FA, and FA components along the critical path shown in bold face in Figure 20. Thus, $\text{T}_{\text{DD}} = \text{TRFD}_1 + \text{TD}_1 + \text{TRFN}_1 + \text{TN}_1 = 2 + 10 + 2 + 10 = 24$. Since the 4x4 multiplier has a longest path delay of 10 threshold gates, then from the flowchart in Figure 14, the 4x4 multiplier can be pipelined with either 5, 4, 3, 2, or 1 gate delays per stage, if completeness of input can be achieved for each such partition.

For a partition of 5 gate delays per stage, 2 stages are required, as shown in Figure 24. The throughput of this 2-stage design is determined to be (14 gate delays)$^{-1}$, as all equations from the pseudocode in Section 3.1.2 yield this same maximum cycle delay. For a partition of 4 gate delays per stage, 3 stages are required, as shown in Figure 25. The first stage only has 3 gate delays, while stage 2 and stage 3 both have 4 gate delays. The throughput of this 3-stage design is determined to be (12 gate delays)$^{-1}$. The equations from the pseudocode in Section 3.1.2 for stage 2, stage 3, and stages 2 and 3 combined all yield this result. For a partition of 3 gate delays per stage, 4 stages are required, as shown in Figure 26. The first stage has 3 gate delays, stage 2 only has 2 gate delays, and stage 3 and stage 4 both have 3 gate delays. The throughput of this 4-stage design is determined to be (10 gate delays)$^{-1}$. The equations from the pseudocode in Section 3.1.2 for stage 1, stage 3, stage 4, and stages 3 and 4 combined all yield this result. For a partition of 2 gate delays per stage, 7 stages are required, as shown in Figure 27. The first stage only has 1 gate delay, while stages 2 through 7 all have 2 gate delays. The throughput of this 7-stage design is determined to be (8 gate delays)$^{-1}$. The equations from the pseudocode in Section 3.1.2 all yield this result, excluding those for stage 1 and the combination of stages 1 and 2.

A partition into a single gate delay per stage cannot be achieved since the completeness of input criterion is unattainable using only one level of logic with a maximum gate fan-in of 4 inputs. This would require inserting a register between the two levels of logic within both the full-adder and half-adder, which would violate the completeness of input criterion upon which they were designed.

4.2 Summary of Multiplier Designs using Full-Word Completion

The maximum throughput when pipelining the 4x4 multiplier using full-word completion was (8 gate delays)$^{-1}$ as attained by the 7-stage design. Table III compares the throughputs attained from Synopsys simulation and shows that the 7-stage design indeed outperforms all other configurations, as expected by comparing the analytically predicted throughputs. This design has a 19% increase in throughput over the next highest throughput from the 4-stage multiplier, and an 83% increase in throughput over the original non-pipelined design. This increase in throughput was achieved at the expense of inserting 6 asynchronous registers along with corresponding completion logic, as dictated by the flowchart of Figure 14. The simulated throughput was obtained by averaging the throughputs resulting from all 256 possible combinations of input pairs.

4.3 Applying Bit-Wise Completion

After traversing the loop of Figure 14 such that *i=0*, the highest throughput design utilized full-word completion. Bit-wise completion was applied to this design as specified by the flowchart. When switching from full-word completion to bit-wise completion the incomplete AND functions had to be replaced with complete AND functions to satisfy the completeness of input criterion over the new completion sets. The resulting design, shown in Figure 28, reduced the completion logic from 2 gate delays to only 1 gate delay for all registers, thus increasing the throughput from (8 gate delays)$^{-1}$ to (6 gate delays)$^{-1}$. From Synopsys simulation throughput was determined to be 0.257 ns^{-1}, an increase of 21% over the design with an identical number of stages using full-word completion. Thus, the 7-stage 4x4 multiplier utilizing bit-wise completion optimizes throughput.

5.0 Conclusion

Since increasingly finer pipelining of the multiplier did not increase the completion delay, the most finely grained pipelined design was optimal. The non-pipelined design (Figure 20) required a maximum register width of 8 bits while the 7-stage pipelined design (Figure 27) required a maximum register width of 16 bits, and $\lceil \mathrm{Log}_4 8 \rceil = \lceil \mathrm{Log}_4 16 \rceil = 2$. However, if the 7-stage design required a maximum register width of 17 bits instead of 16 bits, the throughput for the 7-stage design using full-word completion would have been the same as for the 4-stage design using full-word completion. Thus, the 4-stage design using full-word completion would have been preferable over its 7-stage counterpart, since it would have had less latency. Bit-wise completion would still have had to be performed on the 7-stage design and possibly the 4-stage design to determine the overall optimal throughput design.

Since the GLP methodology successively partitions an N-stage NCL combinational logic design first into 2 stages, then further into as many as *N* stages, it can produce an optimal pipelined NCL system with significantly increased throughput over its original non-pipelined design. The GLP process may also be partially applied to design maximum throughput systems under the constraints of latency and/or area bounds. The GLP methodology combines both full-word completion as well as bit-wise completion for designing the optimal system. A case study of the 4x4 multiplier substantiates the utility and potential for automation of the proposed methodology, as the throughput of the non-pipelined 4x4 multiplier was increased by 125%. In this paper GLP was applied to a dual-rail NCL design; but it can also be applied to a quad-rail NCL design, by inserting quad-rail registers, described in [28], rather than dual-rail registers.

Moreover, although NCL requires both a DATA wavefront and a NULL wavefront, which reduces the maximum attainable throughput by approximately half, a technique can be used to reduce this inherent throughput loss. This *NULL Cycle Reduction* technique [29] exploits parallelism by partitioning input wavefronts such that one circuit processes a DATA wavefront, while its duplicate processes a NULL wavefront. The outputs of the two circuits are then multiplexed to form a single output stream.

References

[1] Karl M. Fant and Scott A. Brandt, "NULL Convention Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis*," International Conference on Application Specific Systems, Architectures, and Processors*, pp. 261-273, 1996.

[2] Ivan E. Sutherland, “Micropipelines,” *Communications of the ACM*, Vol. 32, No. 6, pp. 720-738, 1989.

[3] Paul Day and J. Viv. Woods, “Investigation into Micropipeline Latch Design Styles,” *IEEE Transactions on VLSI Systems*, Vol. 3, No. 2, pp. 264-272, 1995.

[4] K. Yun, P. Beerel, and J. Arceo, “High-Performance Asynchronous Pipeline Circuits,” *Advanced Research in Asynchronous Circuits and Systems*, pp. 17-28, 1996.

[5] Stephen B. Furber and Paul Day, “Four-Phase Micropipeline Latch Control Circuits,” *IEEE Transactions on VLSI Systems*, Vol. 4, No. 2, pp. 247-253, 1996.

[6] J. D. Garside, S. B. Furber, and S. H. Chung, “AMULET3 Revealed,” *Proc. Async ’99*, pp. 51 – 59, 1999.

[7] N.C. Paver, P. Day, C. Farnsworth, D.L. Jackson, W.A. Lien, J. Liu, “A Low-Power, Low Noise, Configurable Self-Timed DSP,” *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 32-42, 1998.

[8] O. Hauck and S. A. Huss, “Asynchronous Wave Pipelines for High Throughput Datapaths,” *IEEE International Conference on Electronics, Circuits, and Systems*, Vol. 1, pp. 283 –286, 1998.

[9] Chansub Park and Duckjin Chung, “Modified Asynchronous Wave-Pipelining,” *Electronics Letters*, Vol. 36, No. 4, pp. 295 –297, 2000.

[10] Jens Sparso and Jorgen Stanstrup, “Design and Performance Analysis of Delay Insensitive Multi-Ring Structures,” *Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences*, Vol.1, pp. 349 –358, 1993.

[11] J. Sparso, J. Staunstrup, M. Dantzer-Sorensen, Design of Delay Insensitive Circuits using Multi-Ring Structures. *Proceedings of the European Design Automation Conference*, pp. 15-20, 1992.

[12] S. Kim and P. A. Beerel, “Pipeline Optimization for Asynchronous Circuits: Complexity Analysis and an Efficient Optimal Algorithm,” *IEEE/ACM International Conference on Computer Aided Design*, pp. 296 –302, 2000.

[13] M. Singh and S. M. Nowick, “High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths,” *Proceeding of the Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 198 –209, 2000.

[14] C. D. Nielsen and A.J. Martin, “Design of a Delay-Insensitive Multiply and Accumulate Unit,” *Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences*, Vol. 1, pp. 379 –388, 1993.

[15] C. L. Seitz, “System Timing,” in *Introduction to VLSI Systems*, Addison-Wesley, pp. 218-262, 1980.

[16] A. J. Martin, “Programming in VLSI,” in *Development in Concurrency and Communication*, Addison-Wesley, pp. 1 – 64, 1990.

[17] K. Van Berkel, “Beware the Isochronic Fork,” *Integration, The VLSI Journal,* Vol. 13, No. 2, pp. 103-128, 1992.

[18] D. E. Muller, “Asynchronous Logics and Application to Information Processing,” in *Switching Theory in Space Technology*, Stanford University Press, pp. 289-297, 1963.

[19] N. P. Singh, *A Design Methodology for Self-Timed Systems*, Master’s Thesis, MIT/LCS/TR-258, Laboratory for Computer Science, MIT, 1981.

[20] 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, pp. 2-10,1992.

[21] T. S. Anantharaman, “A Delay Insensitive Regular Expression Recognizer,” *IEEE VLSI Technology Bulletin*, Sept. 1986.

[22] A. J. Martin, “Compiling Communicating Processes into Delay-Insensitive VLSI Circuits,” *Distributed Computing*, Vol. 1, No. 4, pp. 226-234, 1986.

[23] C. H. (Kees) van Berkel, *Handshake Ciruits: An Intermediary Between Communicating Processes and VLSI*, Ph.D. Thesis, Eindhoven University of Technology, 1992.

[24] Karl M. Fant and Scott A. Brandt, *NULL Convention Logic Systems*, US patent 5,305,463 April 19, 1994.

[25] T. Verhoeff, “Delay-Insensitive Codes – An Overview,” *Distributed Computing,* Vol. 3, pp. 1-8, 1988.

[26] A. Martin, “The Limitations to Delay-lnsensitivity in Asynchronous Circuits,” *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*: pp. 263-278, 1990.

[27] Gerald E. Sobelman and Karl M. Fant, “CMOS Circuit Design of Threshold Gates with Hysteresis,” *IEEE International Symposium on Circuits and Systems* (II), pp. 61-65, 1998.

[28] S. C. Smith, R. F. DeMara, D. Ferguson, and D. Lamb, “Optimization of NULL Convention Self-Timed Circuits,” submitted to *Integration, The VLSI Journal*, August 2001.

[29] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson, "Speedup of Delay-Insensitive Digital Systems Using NULL Cycle Reduction," *The 10th International Workshop on Logic and Synthesis*, pp. 185 – 189, 2001.

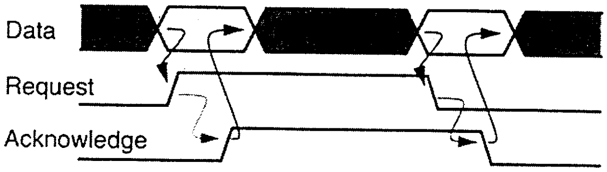

Figure 1. Two-phase handshaking protocol [2].

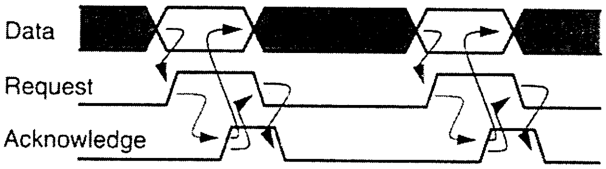

Figure 2. Four-phase handshaking protocol [5].

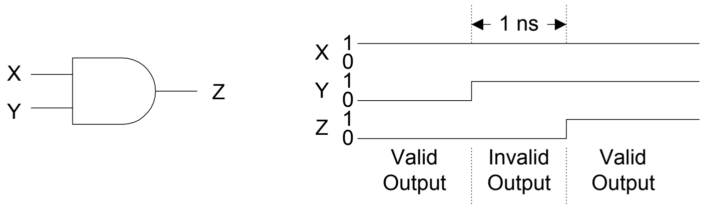

Figure 3. Symbolic incompleteness of a Boolean AND gate.

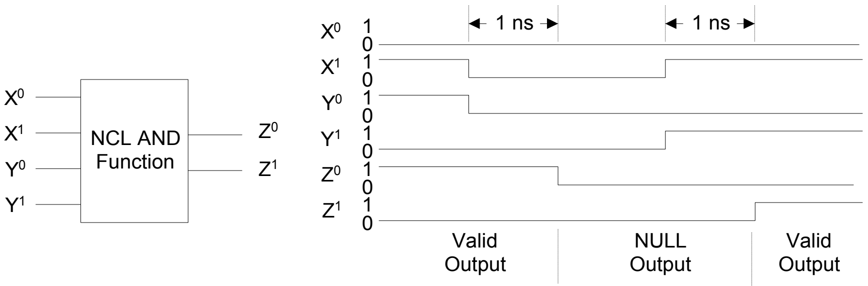

Figure 4. NCL AND function: Z = X • Y and associated waveforms.

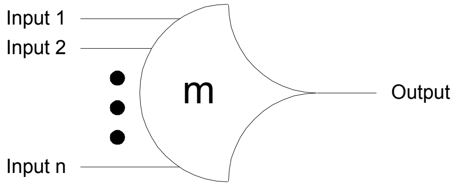

Figure 5. THmn threshold gate [27].

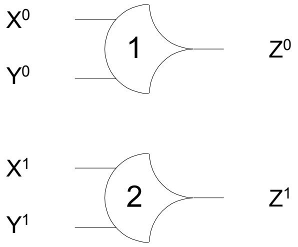

Figure 6. Incomplete AND function: Z = X • Y.

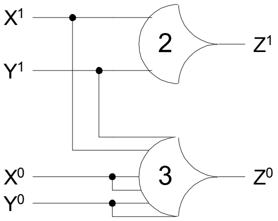

Figure 7. Input-complete AND function: Z = X • Y.

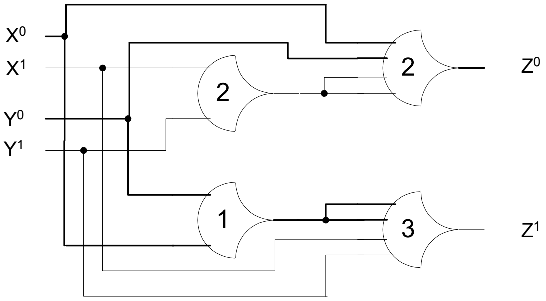

Figure 8. Incorrect XOR function: Z = X ⊕ Y
(orphans may propagate through a gate).

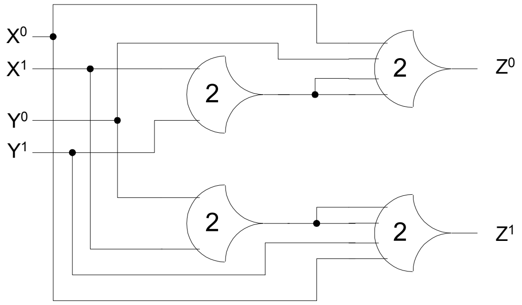

Figure 9. Correct XOR function: Z = X ⊕ Y
(orphans may not propagate through any gate).

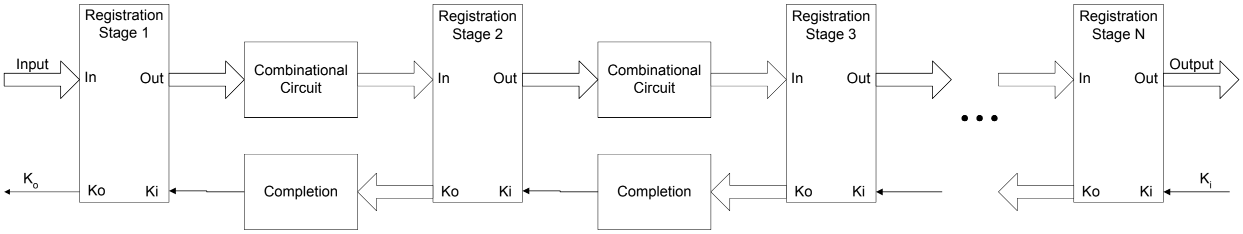

Figure 10. General NCL pipeline [1].

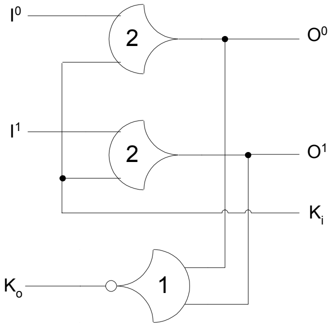

Figure11. Single-bit dual-rail NCL register [1].

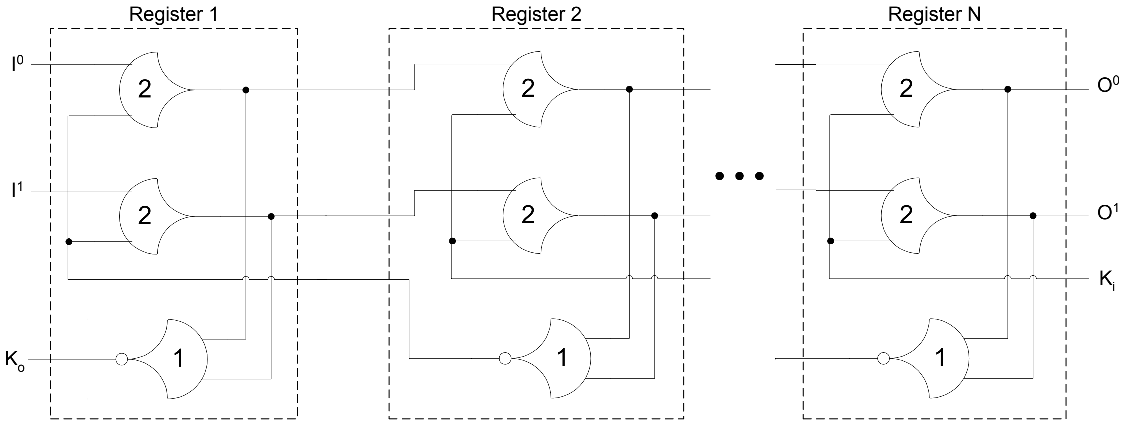

Figure12. Single-bit NCL pipeline.

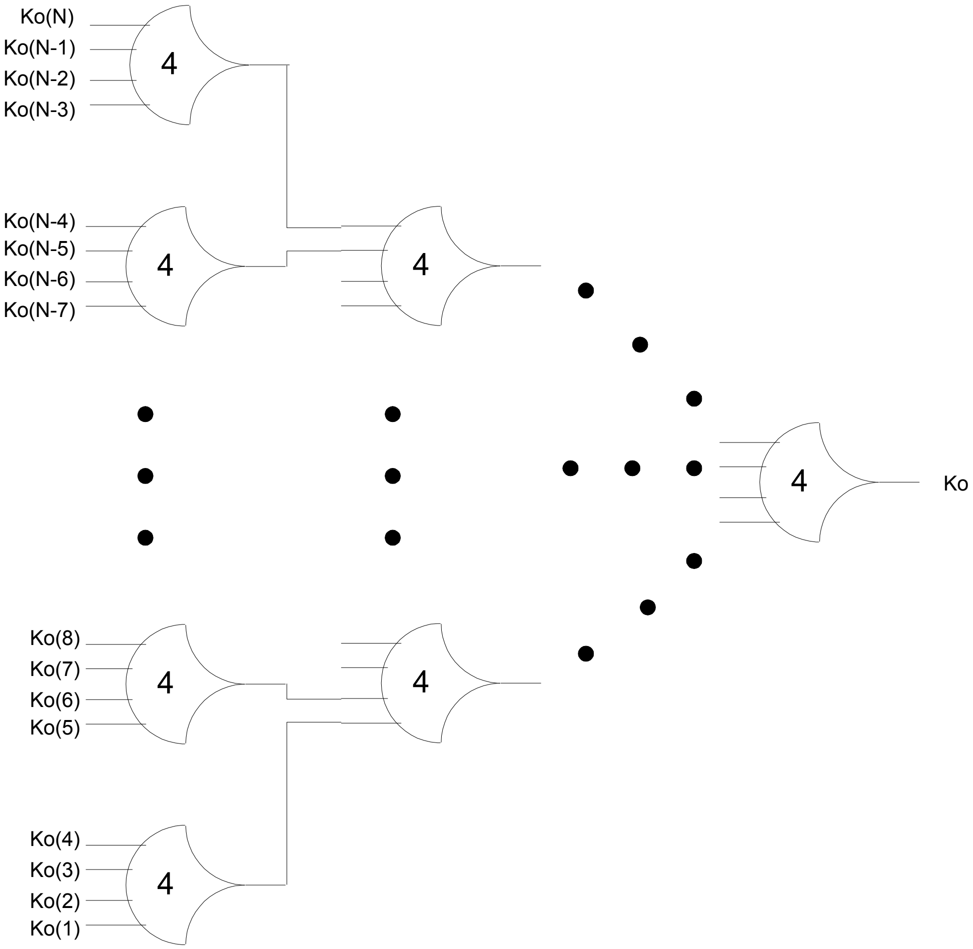

Figure 13. N-bit completion component.

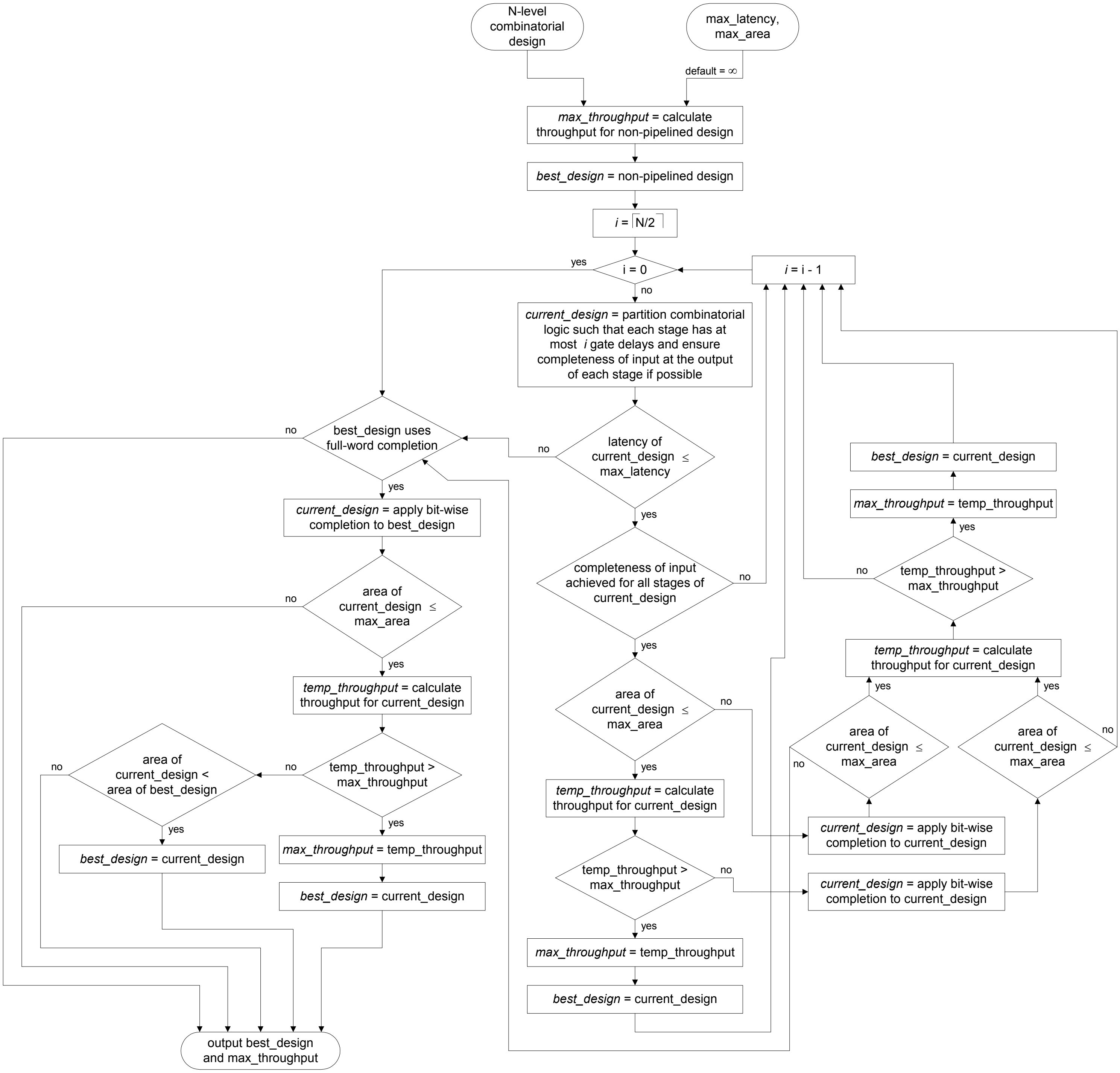
N-level combinatorial design
max_latency, max_area
default = ∞
max_throughput = calculate throughput for non-pipelined design
best_design = non-pipelined design
i = ⌈N/2⌉
i = 0
yes
no
i = i - 1
current_design = partition combinatorial logic such that each stage has at most i gate delays and ensure completeness of input at the output of each stage if possible
latency of current_design ≤ max_latency
completeness of input achieved for all stages of current_design
area of current_design ≤ max_area
temp_throughput = calculate throughput for current_design
temp_throughput > max_throughput
max_throughput = temp_throughput
best_design = current_design
current_design = apply bit-wise completion to current_design
current_design = apply bit-wise completion to current_design
area of current_design ≤ max_area
area of current_design ≤ max_area
temp_throughput = calculate throughput for current_design
temp_throughput > max_throughput
max_throughput = temp_throughput
best_design = current_design
best_design uses full-word completion
current_design = apply bit-wise completion to best_design
area of current_design ≤ max_area
temp_throughput = calculate throughput for current_design
temp_throughput > max_throughput
area of current_design < area of best_design
best_design = current_design
max_throughput = temp_throughput
best_design = current_design
output best_design and max_throughput

Figure 14. GLP methodology design flow.

DATA Combinational Evaluation	DATA Completion Acknowledgement	NULL Combinational Evaluation	NULL Completion Acknowledgement
D	RFN	N	RFD

◄ DATA-to-DATA Cycle ►

Figure 15. Sub-cycles of the NCL cycle.

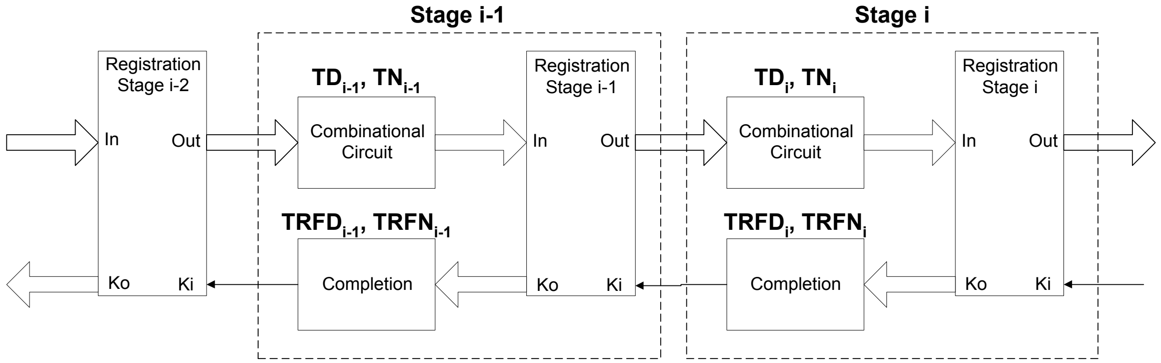

Figure 16. Pipeline showing NCL sub-cycle times.

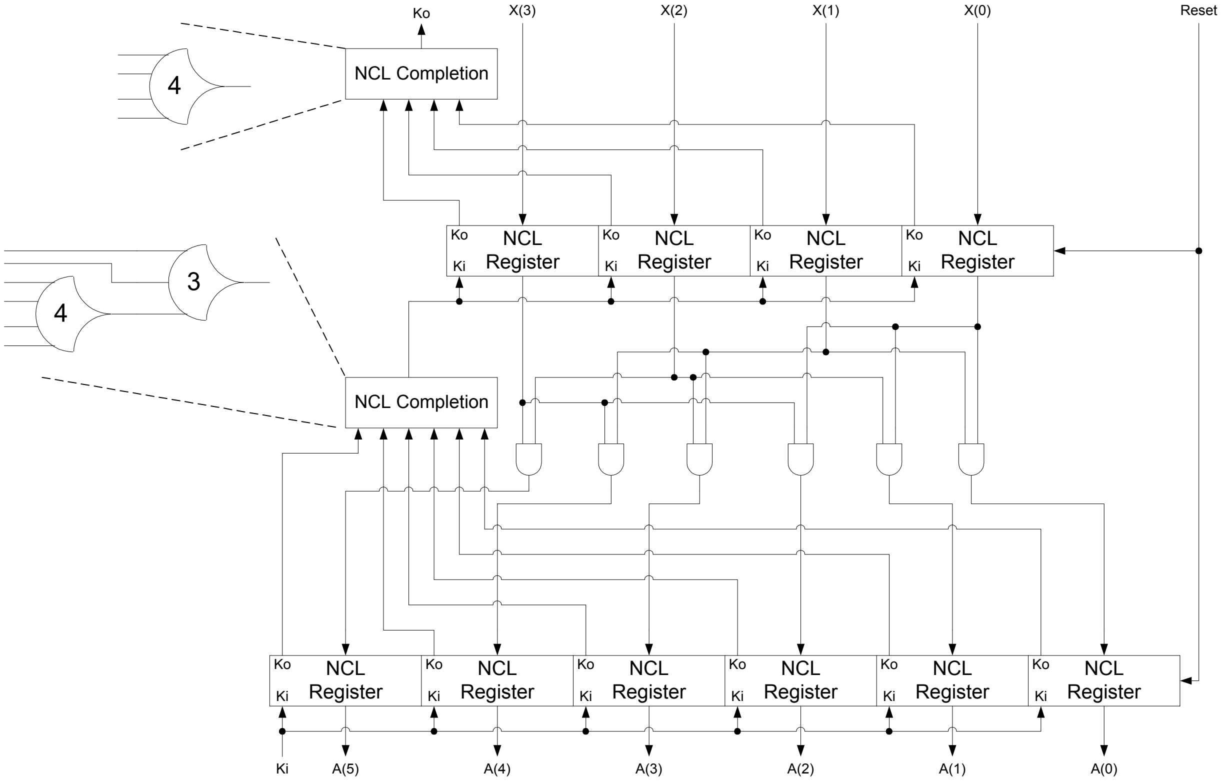

Figure 17. Full-word completion.

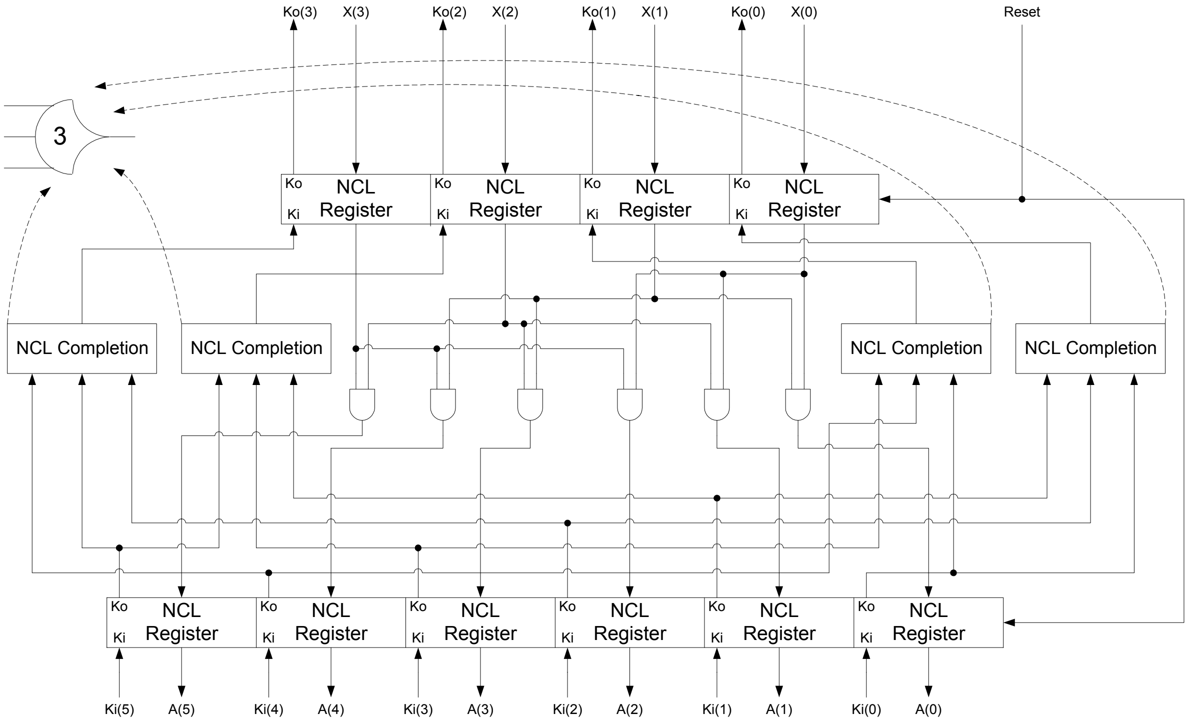

Figure 18. Bit-wise completion.

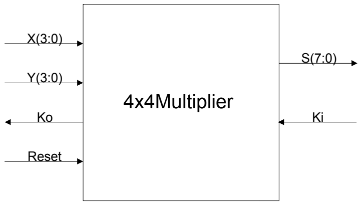

Figure 19. 4x4 multiplier block diagram.

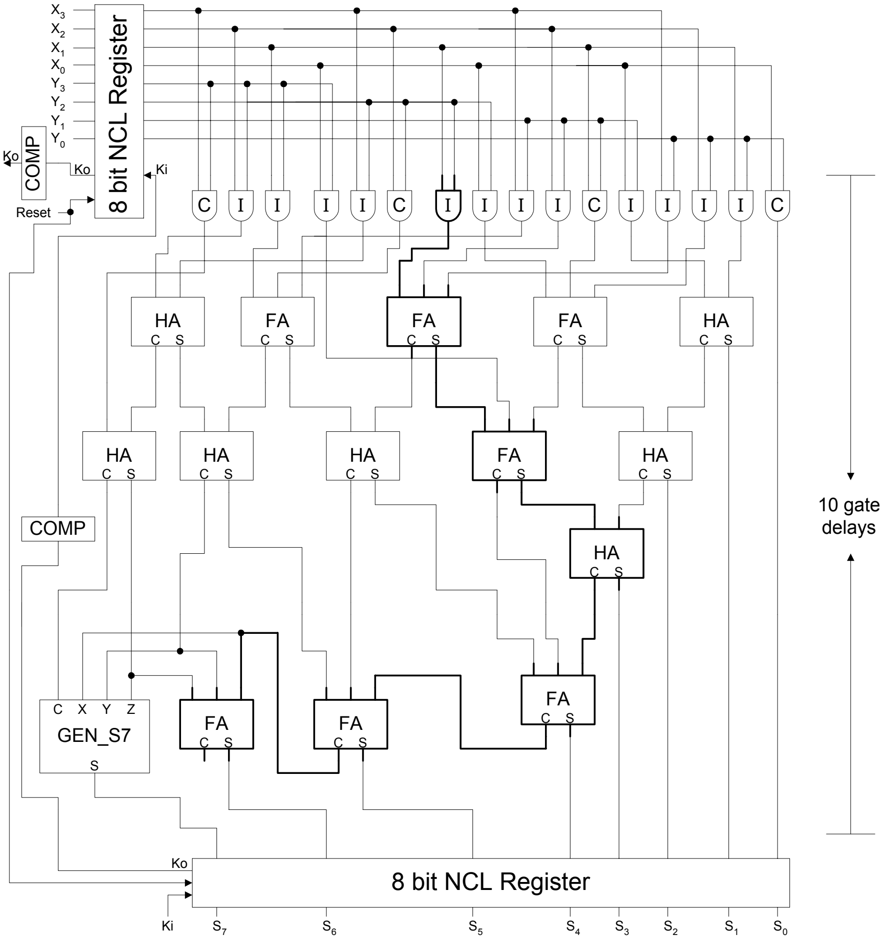

Figure 20. 1-stage 4x4 multiplier using full-word completion.

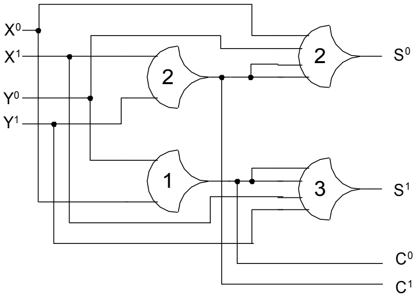

Figure 21. Half-adder.

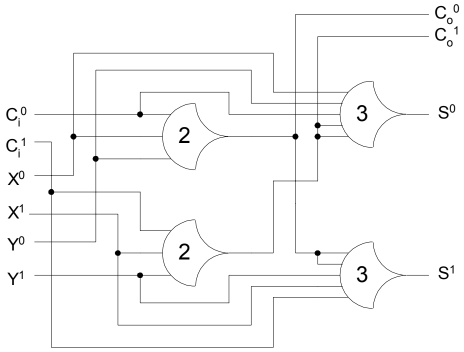

Figure 22. Full-adder.

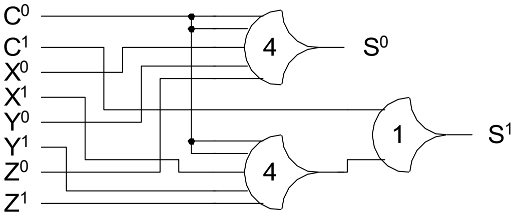

Figure 23. GEN_S7 component.

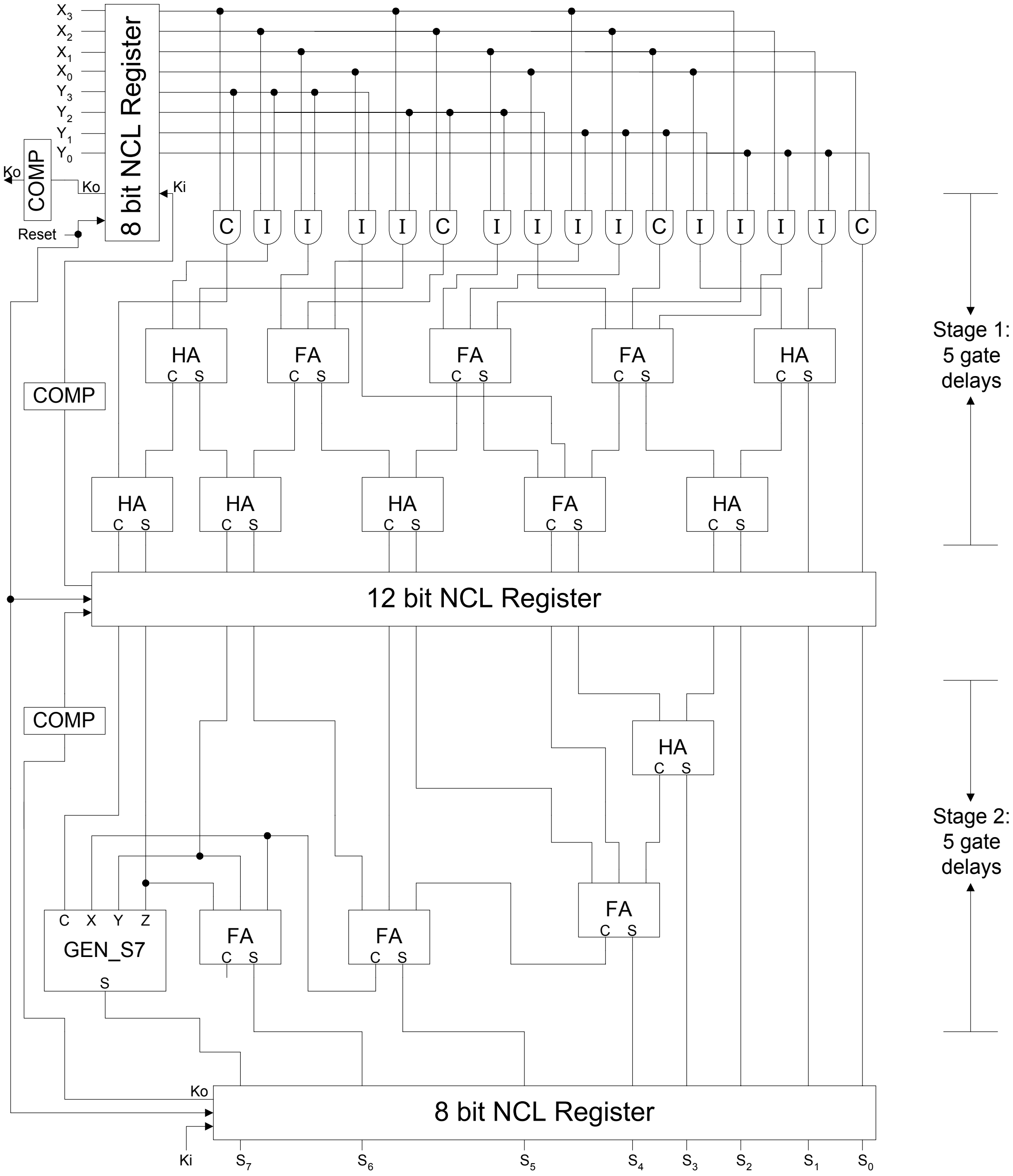

Figure 24. 2-stage 4x4 multiplier using full-word completion.

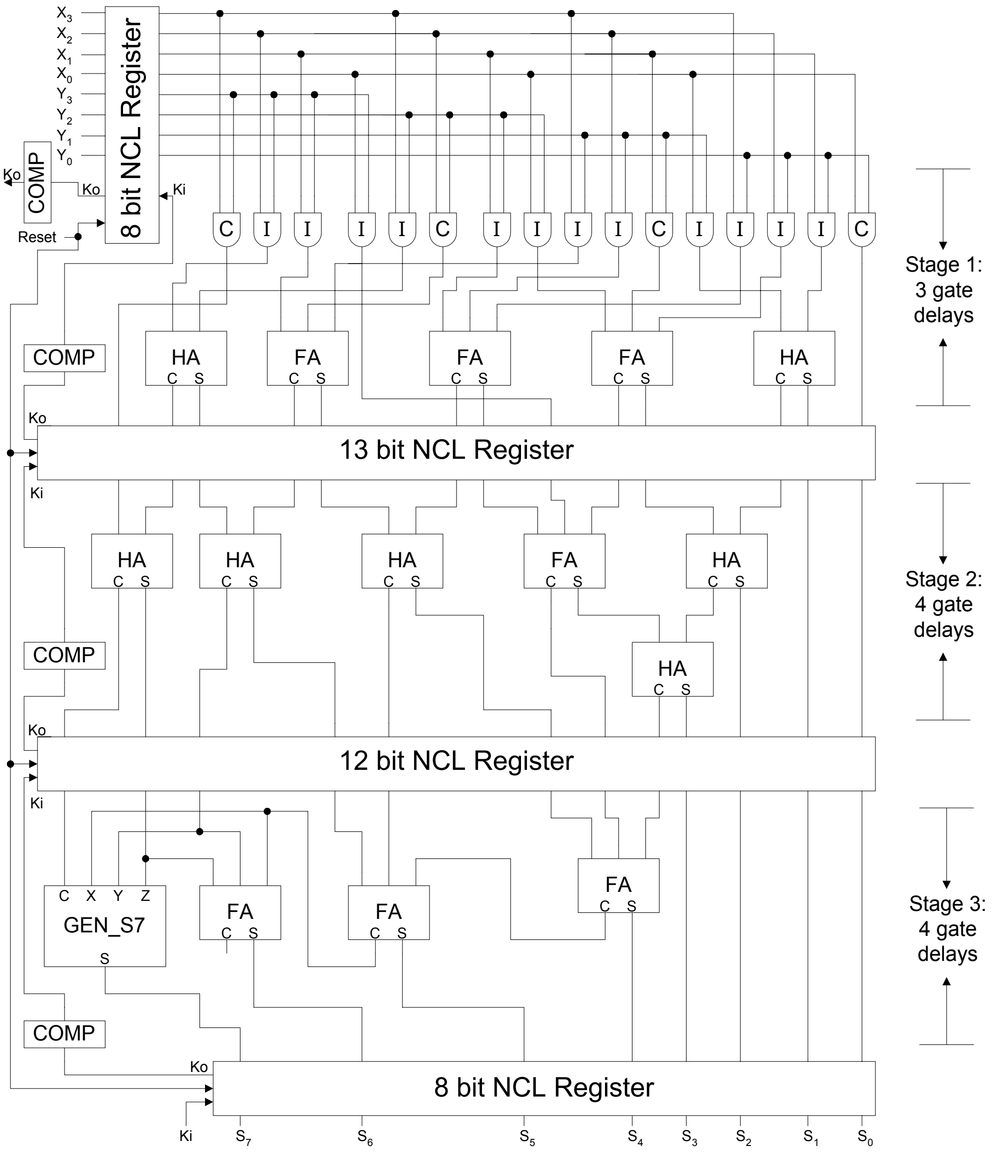

Figure 25. 3-stage 4x4 multiplier using full-word completion.

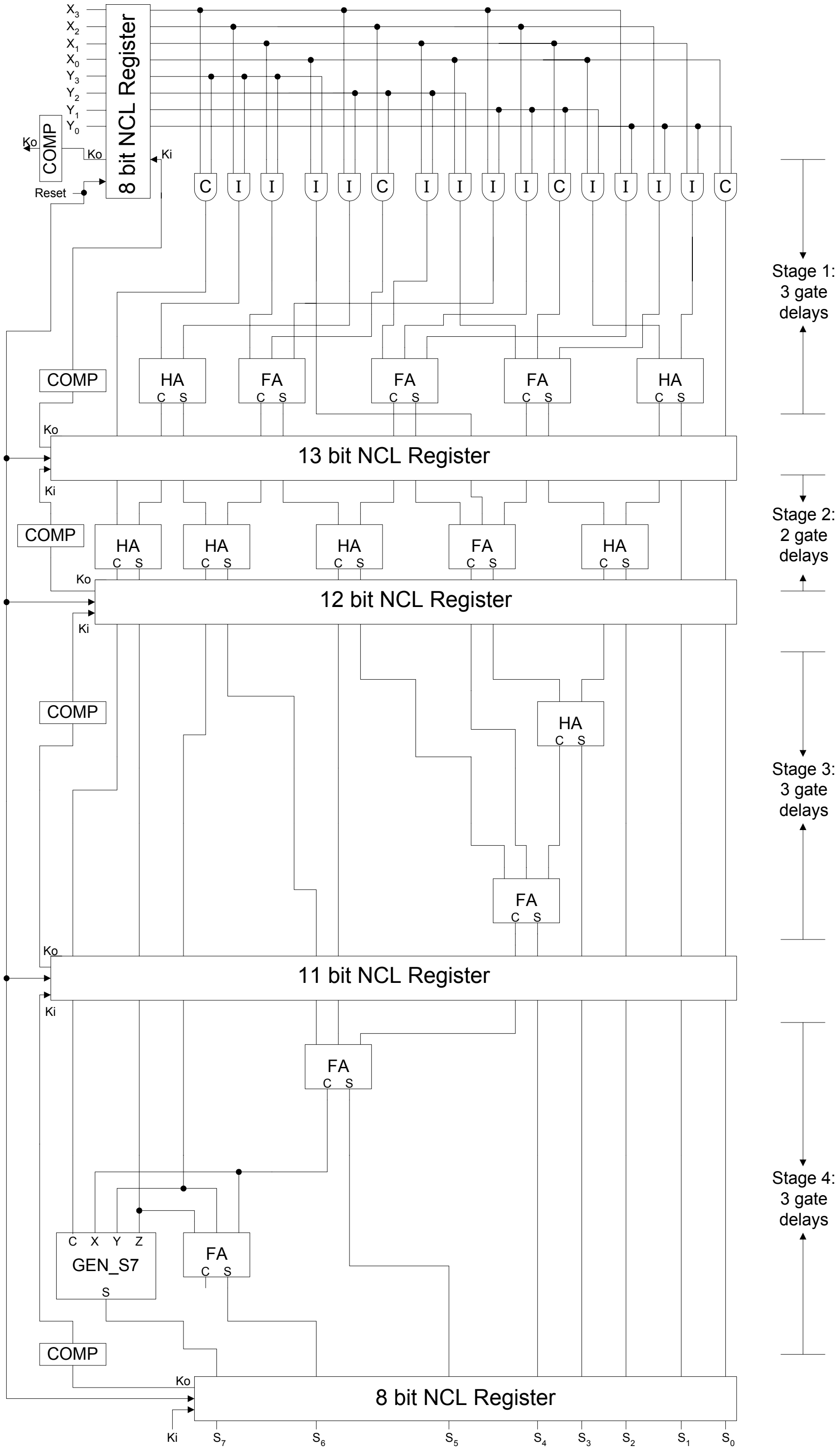

Figure 26. 4-stage 4x4 multiplier using full-word completion.

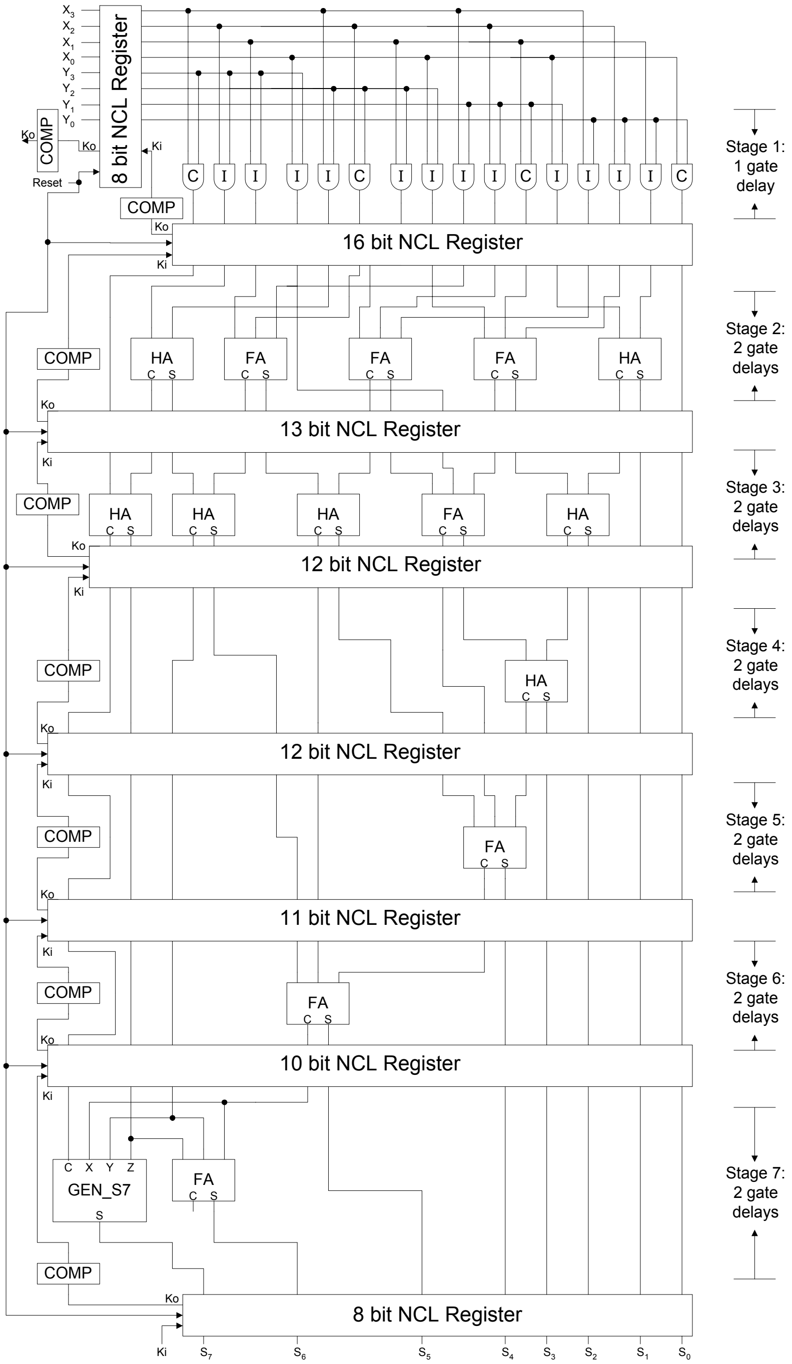

Figure 27. 7-stage 4x4 multiplier using full-word completion.

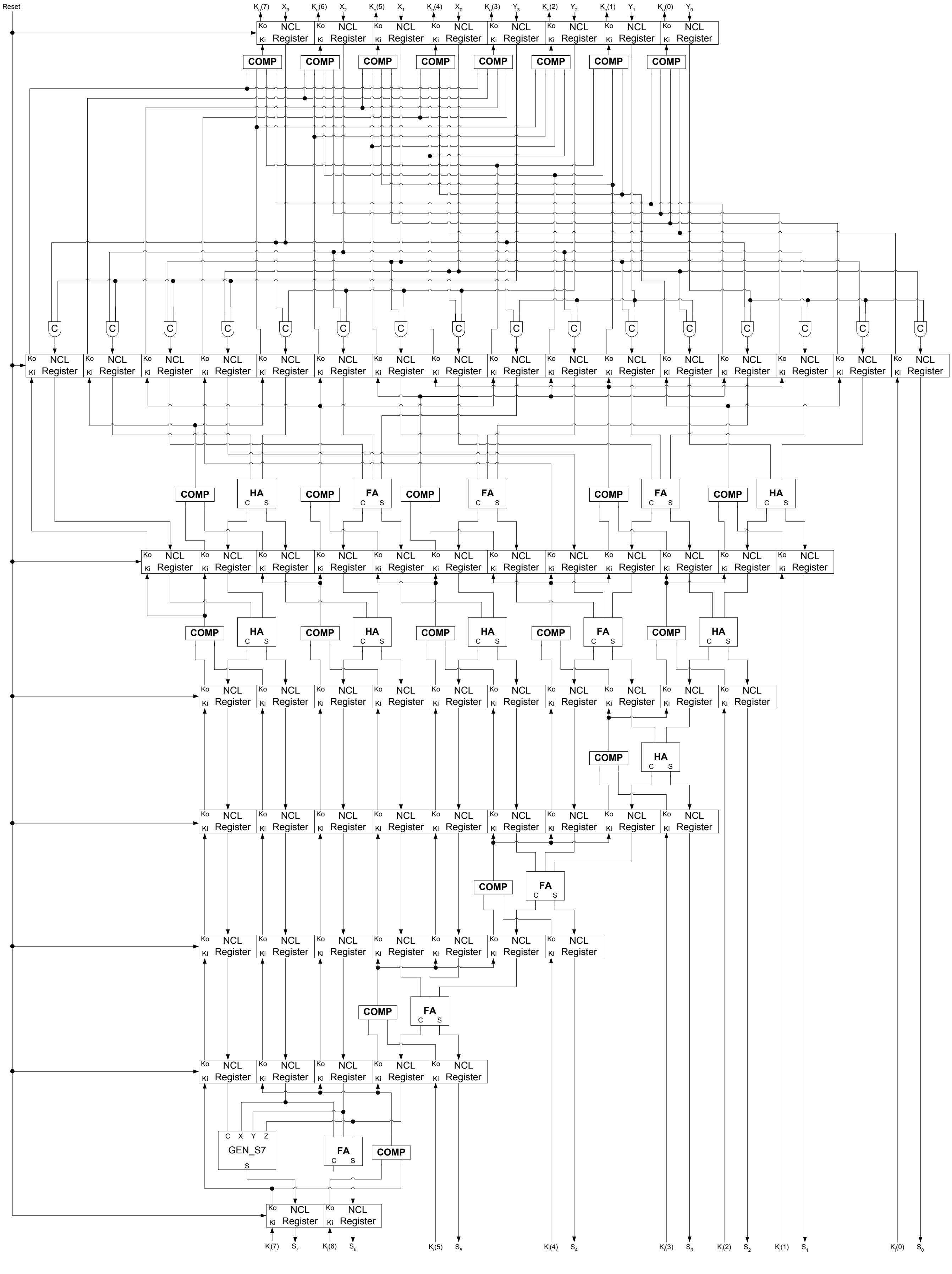

Figure 28. 7-stage 4x4 multiplier using bit-wise completion.

Table I: Discrete timing chart for the idealized NCL cycle.

	Sub-	**Initial**	**Wavefronts**				
Stage	**cycle**	**State**	1	**2**	**3**	4	5
i-1	$\mathbf{D_{i-1}}$		X		**X**		X
	$\mathbf{N_{i-1}}$	X		**X**		X	
i	$\mathbf{D_{i}}$			**X**		X	
	$\mathbf{N_{i}}$	X			**X**		X

Table II: Discrete timing chart for the general NCL cycle.

	Sub-	**Initial**	**Wavefronts**								
Stage	**cycle**	**State**	1	**2**	**3**	**4**	**5**	6	7	8	9
i-1	$\mathbf{D_{i-1}}$		X				**X**				X
	$\mathbf{N_{i-1}}$	X			**X**				X		
	$\mathbf{RFD_{i-1}}$	X				**X**				X	
	$\mathbf{RFN_{i-1}}$			**X**				X			
i	$\mathbf{D_{i}}$			**X**				X			
	$\mathbf{N_{i}}$	X				**X**				X	
	$\mathbf{RFD_{i}}$	X					**X**				X
	$\mathbf{RFN_{i}}$				**X**				X		

Table III: Multiplier comparisons.

Multiplier Design	**Maximum Combinational Delay per Stage** (gate delays)	**Maximum Completion Delay per Stage** (gate delays)	**Predicted Throughput** (gate delays)$^{-1}$	**Simulated Throughput** (ns)$^{-1}$
1-stage	10	2	1/24 = 0.042	0.114
2-stage	5	2	1/14 = 0.071	0.150
3-stage	4	2	1/12 = 0.083	0.172
4-stage	3	2	1/10 = 0.100	0.176
7-stage	2	2	1/8 = 0.125	0.209