Testing of Asynchronous Designs by "Inappropriate" Means. Synchronous approach

Alex Kondratyev

Cadence Berkeley Laboratory
2001 Addison Street, 3rd Floor
Berkeley, CA 94704, USA
kalex@cadence.com

Lief Sorensen, Amy Streich

Theseus Logic, Inc.
485 Keller Road, Suite 140
Maitland, FL 32751, USA
lief@theseus.com, amy@theseus.com

Abstract

The roadblock to wide acceptance of asynchronous methodology is poor CAD support. Current asynchronous design tools require a significant re-education of designers, and their capabilities are far behind synchronous commercial tools. This paper considers the testing methodology for a particular subclass of asynchronous circuits (Null Convention Logic or NCL) that entirely relies on conventional CAD tools available at today's market. It is shown that for acyclic NCL pipelines a test pattern generation for stuck-at faults could be effectively solved through the construction and checking of the synchronous circuit with a set of faults "equivalent" to the original NCL circuit. This result is extended to arbitrary NCL structures by applying the partial scan technique to break computational loops. The method guarantees 100% stuck-at fault coverage in NCL systems, which is confirmed by experimental data.

1. Introduction

Asynchronous designs have been proven capable of delivering:

- Higher speed because of the average case performance versus worst case in synchronous circuits [1, 2]
- Less power consumption due to the absence of clock trees and natural support of idle mode [3, 4]
- Low EMI and noise due to even distribution of switching activity in time [5, 6]

Nevertheless, the few success stories did little to change the public acceptance of asynchronous approaches. In the current design landscape, asynchronous methodologies are considered either as exotic (an optimistic opinion from academia) or as non-observable (a pessimistic opinion from industry). The blame is partially on the asynchronous community because for several decades it persistently ignored the trends and standards in synchronous methodologies which are the mainstream for the rest of the world. Most asynchronous approaches use particular non-conventional models (Burst-mode machines [7,8], Communicating Processes [9,10], Signal Transition Graphs [11,12], etc.) which are supported by custom in-house tools. This leaves few hopes to compete with synchronous methodologies that are supported by the full power of EDA industry.

Recently, the ideas of using commercial CAD tools and conventional HDLs with an asynchronous approach have gained much interest. In [13] a tool for automated translation of CSP-like specifications into VHDL programs was developed. Blunno and Lavagno [14] suggested a compiler from a RTL description in Verilog to an asynchronous controller and a synchronous datapath supporting the micro-pipeline design style. The closest to our work [15] has presented a new synthesis framework for the design of Null Convention Logic (NCL) circuits [16] from VHDL specifications using commercial CAD tools. These works have made an important step in covering the gap between the ease of use of asynchronous and synchronous approaches. Providing conventional synthesis and simulation capabilities in an asynchronous design flow removes the roadblock of reeducating the designers and shifts the choice of "usage/non-usage" of asynchronous circuits into an objective ground of estimating their trade-offs: area, speed, power, etc.

Note however, that synthesis and simulation address only part of the design problems. A methodology cannot be complete if it does not provide means for design testing. There is no common agreement about the comparative complexity of asynchronous vs synchronous testing. On one hand, many asynchronous circuits enjoy the well-known self-checking properties for output stuck-at faults [12,17]. On the other hand, asynchronous systems tend to have more sequential gates and are difficult to pause, thus creating a more complex test environment. The brute force approach, based on an exhaustive exploration of the reachability space, shows that asynchronous circuits might have high potential test coverage [18] but its application is limited to circuits of very moderate size. Restricting implementation choices allows a designer to use more efficient ad hoc techniques. Micro-pipelined architectures, [19,20] provide regular methods to ensure high coverage of stuck-at and delay faults. Testability in these methods comes at the expense of a structural modification of asynchronous latches (C-elements) and might be costly.

Another effective test approach is suggested in [21,22] for the design of handshake circuits. It covers both structural modifications of handshake components and a partial scan application to increase the testability of designs. However, the methodology as a whole is based on a non-conventional specification means (Tangram language) and in-house CAD tools (Tangram compiler), which is an obstacle in accepting it externally.

This paper targets the development of test methods that support NCL design flow. The main distinctive feature of the suggested approach is the use of conventional Automatic Test Pattern Generation (ATPG) tools to generate test vectors for NCL circuits. The key is to substitute inside the ATPG engine an original NCL system by a synchronous circuit with an "equivalent" to NCL fault set. It is proved that for acyclic pipelines the ATPG approach guarantees detection of **all** input stuck-at faults in a NCL circuit. Together with the application of partial scan techniques [26] (to break up computational loops) it shows that NCL systems are fully testable with respect to stuck-at faults.

The paper is organized as follows. Section 2 introduces the main theoretical notions. Section 3 discusses the ATPG setting for testing acyclic NCL pipelines and then extends the approach for arbitrary NCL structures. Section 4 presents experimental results for the suggested flow.

2. Theoretical Background

2.1 NCL Combinational Circuits

NCL circuit inputs and outputs use a **delay-insensitive encoding** [19]. A circuit assumes a two-phase functioning in which data communication alternates between **set** and **reset phases** [20]. Data changes from the spacer (**NULL**) to a proper codeword (**DATA**) in the set phase, and then back to NULL in the reset phase.

In NCL this behavior is pushed down to the level of each particular gate of a circuit. If the current state of a gate is NULL, then the gate keeps its output in NULL while NULL is present in at least one of its fan-ins. When **all** gate fan-ins receive a codeword (DATA), the output of the gate changes to DATA. A gate has a symmetric behavior in the reset phase – it keeps output in DATA until **all** the fan-ins receive NULL; after which the output changes to NULL. It is easy to see that such gates guarantee the delay-insensitivity of implementation because the gates' outputs acknowledge any changes at their inputs.

This behavior is naturally expressed in a multi-value logic. Let a signal in NCL take three logic values: data values "1" and "0" and the value "N" (NULL). The behavior of basic NCL gates is described like in Figure 1(a) (gates are assumed to be initially in a state NULL). The description of behavior of basic NCL gates is accomplished by symbolic tables for initial state DATA (see Figure 1(b), where H stands for holding the previous DATA state of the gate (1 or 0) while one of the inputs changes to NULL).

From the above explanation it follows that NCL gates have sequential behavior because they switch differently depending on the current value on the output.

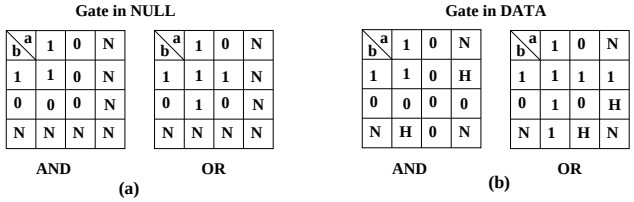

Figure 1. Symbolic tables for basic NCL gates

For physical implementations of 3-value gates, each logical signal *a* is represented by two wires *a.1* and *a.0* resulting in a well-known dual-rail encoding (see left portion of Table 1).

a	*a.0, a.1*	*a*	*a.0,a.1,a.2,a.3*
0	1 0	0	1 0 0 0
1	0 1	1	0 1 0 0
N	0 0	2	0 0 1 0
		3	0 0 0 1
		N	0 0 0 0

Table 1 . One-hot encodings

In general, NCL is not restricted to dual-rail implementations only. Other schemes based on one-hot encoding are used as well. In the right portion of Table 1 a 4-rail encoding of 5-value NCL (data values "0","1",'2","3" and NULL value "N") is shown. This encoding is known to be beneficial for low-power implementations [23].

A representation of sequential NCL gates through set *S* and reset *R* functions ($g=S+gR^{\wedge}$) might be refined based on the following specific features of NCL behavior and one-hot encoding with a spacer "000…":

1) in a multi-rail circuit a transition from NULL to DATA is monotonic
2) a transition from DATA to NULL at primary inputs sets all gates in a combinational circuit to NULL state

From (1) it follows that a set function *S* of a gate must be **positively unate**, i.e. every variable is met in function *S* without inversion.

(2) concludes that an NCL gate changes its output to NULL when all its inputs are at NULL. Since DATA values are one-hot encoded we arrive at:

$$R^{\wedge}(x_1,\ldots,x_n) = x_1 + x_2 + \ldots + x_n$$

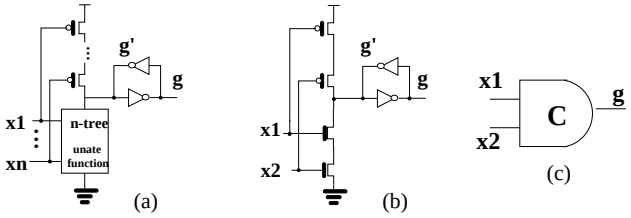

Figure 2. NCL gate implementations

A semi-static CMOS implementation of an NCL gate is shown in Figure 2(a), while Figure 2(b)(c) show an implementation and notation for a particular NCL gate with the function $g=x1x2 + g(x_1+x_2)$, known from literature as a Muller's C-element.

Breaking a single logical signal *a* into several physical instances *a.0*, *a.1*, *a.2* , ... under a one-hot encoding in an implementation, might destroy the delay-insensitive properties which multi-value NCL gates naturally provide. In an automated design flow, delay-insensitivity can be ensured by using correct-by-construction synthesis methods [15]. For manual designs, delay-insensitivity should be verified by checking that primary outputs acknowledge every transition of an internal gate or input. If some gates or wires of a circuit are unable to translate the results of their firings into the changes at primary outputs, a circuit is said to contain so-called **orphans** [24]. An orphan can be characterized by a path *p* in a circuit such that a transition propagates through *p* but its propagation does not affect output values. If *p* consists of a single wire then it is called a **wire orphan**. Orphan analysis is a part of NCL design flow and the method of its efficient implementation is described in a companion paper. From now on we assume that a properly designed NCL circuit has no orphans other than wire orphans.

2.2 NCL Systems

At an architectural level, NCL systems show a clear separation of sequential and combinational parts, much *in the same way as with synchronous systems*.

Figure 3 (a) shows a structure of a NCL dual-rail register. The register consists of transparent latches implemented by C-elements, with a common enabling signal *Req*. Data inputs to the register come in a dual-rail form from the preceding combinational logic while data outputs are forked to the next combinational logic and to a completion detector (see Figure 3(b)). For a one-hot encoded set of wires *a.0*, *a.1*, *a.2*, ... only one of the wires can take the value "1" in a set phase, while in a reset phase all wires will take values of "0". From this it follows that to detect a proper codeword (or a spacer) at *a.0*, *a.1*, *a.2*, ... one can simply use an OR gate. The common completion detector for the register combines the detectors for each bit (OR gates) through a C-element (see Figure 3(b)).

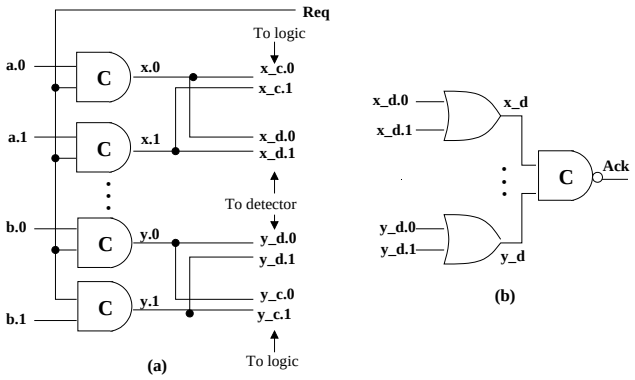

Figure 3. NCL register (a) and its completion detector (b)

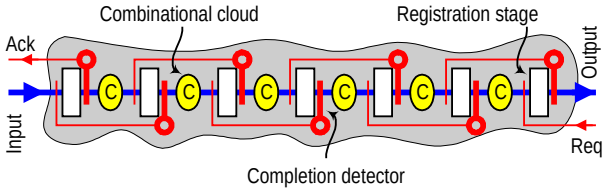

Figure 4. NCL system

The general structure of a NCL system is shown in Figure 4. To explain how the NCL system functions, let us assume that all registers are initially in the NULL state. These register states are detected by **completion detectors,** and all **acknowledgement signals** *Ack* are asserted to "1". The latter asserts the **request signals** (*Req*) and prepares the registers for accepting DATA codewords on their inputs. When DATA arrives, the outputs of a register transition from NULL to DATA (the register stores the DATA value), and the DATA **wavefront** propagates through a combinational circuit to the inputs of the next register. Simultaneously, a completion detector checks for a DATA codeword at its inputs, and replies by de-asserting the *Ack* signal. This signal disables the request line of the previous register and prepares the register for storing the next NULL wavefront. The **request-acknowledgement mechanism** of register interaction [12] ensures a **two-phase discipline** in NCL system functionality and prevents collisions between different DATA wavefronts (any two DATA wavefronts are separated by a NULL wavefront between them).

3. Test Pattern Generation for NCL systems

3.1 Type of faults

Testing procedures most commonly target the following types of faults: a) stuck-at faults b) delay faults c) bridging faults.

Stuck-at faults are modeled by assigning a fixed 0 or 1 value to a circuit gate input or output. In this case, a gate is considered as a "black box" which allows a designer to separate a gate's functionality from its technology dependent implementation. Checking for stuck-at faults in NCL circuits is fully addressed by the suggested approach. Note, that due to considering gates as atomic instances the faults in internal feedbacks of NCL gates (node *g'* in Figure 2) are not checked. We see the significance of this limitation in the advocated approach and put the task of testing the internal feedbacks as an important direction for future investigation.

Delay-insensitive NCL systems are immune to timing failures. Delay faults in these circuits influence the circuit performance but not the correctness of behavior. However, testing for delay faults in NCL circuits might still be meaningful due to two reasons 1) from a practical point of view a circuit whose response time is arbitrary has little use and 2) other types of faults (beyond stuck-at) could show up as delay faults. Though the presented approach is restricted to stuck-at faults only the suggested framework might be useful in delay fault testing as well. Here one can exploit the idea of checking delay faults through the reduction to stuck-at faults testing which is well known in synchronous methodology [25]. This issue is however beyond the scope of this paper.

For testing bridging faults, stuck-shorts and stuck-opens our method provides no help. These faults might be approached by applying an IDDQ technique which is orthogonal to the suggested ATPG-based approach.

3.2 Acyclic Pipelines

Let us first confine ourselves by considering NCL systems that are free from computational loops. These ssytems are represented by acyclic pipelines, where data travels from primary inputs (PI) to primary outputs (PO) as in Figure 4 (the pipeline should not necessarily be linear but may contain arbitrary forks and joins).

During testing we assume that an acyclic pipeline is controlled only from PI and is observed only at PO, i.e. no additional controllable or observable points are added to the pipeline and testability is checked under zero area overhead.

In general a designer might choose test vectors that are never propagated in normal circuit functionality. However, for NCL pipelines, using valid DATA patterns (respecting one-hot encoding) and following the conventional NULL-DATA protocol is preferable for two reasons:

1. Ease of propagation (if test vectors follow DATA-NULL convention, they are easily propagated through the system since setting the PI of a fault-free pipeline to DATA (NULL) eventually sets the PO to DATA (NULL) while propagation is not guaranteed with arbitrary vectors)
2. Speed of testing (a pipeline can store several DATA and NULL items which are safely separated by pipeline stages and never interfere)

Therefore, the suggested approach suggests a test procedure that applies valid DATA/NULL pairs at the PI of a pipeline and observes the corresponding DATA/NULL pairs at the PO of a pipeline.

3.2.1 Faults in pipeline registers

The acyclic pipeline contains registers and feedbacks due to request-acknowledge interaction and therefore combinational ATPG methods cannot be applied directly. However, the problem can be simplified by using fault grading.

Faults are separated according to the location of their occurrence. Stuck-at faults in the register include:

1. Enabling lines (wire *Req* and its forks in Figure 3)
2. Register outputs towards the completion detector (wires *x_d.0* and *x_d.1* e.g. in Figure 3)
3. Internal nodes and the output of the completion detector (wires *x_d* , *y_d* and *Ack* in Figure 3)

The remaining faults are referred to as faults in combinational logic.

In a fault-free NCL pipeline all internal nodes and outputs of completion detectors toggle in each phase. A stuck-at fault at any internal node of a completion detector freezes its output and eventually stalls the pipeline since data propagation through the register is possible only upon the consistent switching of data and enabling lines (switching to "1" for set phase and to "0" for reset).

Proposition 1. For an acyclic NCL pipeline with *M* stages, checking the faults in all completion detectors can be performed by applying at most *M/2+1* (*NULL*,*DATA*) input patterns.

Proof. Suppose that the pipeline is in a settled but unknown initial state. The NCL pipeline of *M* stages might have at most *M/2 DATA* items (because each *DATA* item is separated by a stage in the *NULL* state). Let us perform *M/2+1* handshakes at the input of a pipeline. Every time primary outputs *PO* settle to valid *DATA* or *NULL*, *Req* is changed to prepare for the next wavefront (*NULL* or *DATA*) propagation. If the output handshake is performed *M/2+1* times, then it is guaranteed that at least one *DATA* and one *NULL* wavefront will have propagated all the way through from *PI* to *PO*. The latter means that every

completion detector has toggled to "1" and "0" and is free from stuck-at faults. ♦

Note that the test procedure referred to by Proposition 1 does not depend on the particular value of *DATA* pattern but relies only on the alternation of wavefronts at pipeline inputs. In fact the same *DATA* item might be repeated *M/2+1* times. The latter shows that stuck-at faults in completion detectors are easy to check and this circuitry might be dropped from further consideration.

The rest of the faults in the registers could be eliminated by fault collapsing based on dominance relation [26].

The faults at enabling lines and their forks are dominated by faults at data inputs to registers. If a test vector checks the propagation of a rising (falling) transition at data input *a* through a latch *x* of a register, then it also checks that the corresponding rising (falling) transition has properly occurred at the enabling line of *x*.

Faults at register outputs connected to completion detectors and to combinational logic are also related. Indeed, suppose that data output from register latch *x* forks to *x_c* connected to the next stage logic and *x_d* connected to completion detector (see e.g. wires *x_c.0* and *x_d.0* in Figure 3(a)). For checking a stuck-at-0 (stuck-at-1) fault at *x_c*, a rising (falling) transition on *x_c* must be propagated through the pipeline. In a faulty circuit this transition does not propagate and the pipeline stalls. It is easy to see that application of the same transition will stall the pipeline for a stuck-at fault at *x_d* as well because the completion detector for *x* will not switch.

From the above discussion, it follows that register circuitry can also removed from consideration when generating test patterns for NCL pipelines. Intuitively this is easy to understand because transparent latches under monotonic changes at data inputs behave like buffers and do not influence testability of a system. Therefore, for the purpose of test pattern generation one might take a simplified view of the NCL pipeline as a connection of combinational logic networks in sequence, which is equivalent to a **single** combinational logic network (see Figure 5).

3.2.2 Faults in combinational logic

Checking stuck-at 1 and 0 faults is performed by propagating falling and rising transitions through a circuit respectively. From the rules of functionality of a NCL combinational circuit, it follows that rising (falling) transitions at gate outputs occur only in a set (reset) phase. Therefore, if the testing is performed by applying (DATA, NULL) testing patterns at primary inputs, stuck-at-0 faults would be checked in a set phase while stuck-at-1 faults would be checked during a reset phase.

The two-phase operation of NCL circuits also helps in simplifying the analysis of a network of sequential NCL gates. Let us assume that before applying a new test pattern, a NCL network is in a settled state. Then it is easy to see that the behavior of sequential NCL gates $g(x_1,\ldots,x_n) = S + g(x_1+x_2+\ldots+x_n)$ is specified by their set functions S for set phase and by their reset functions $x_1+x_2+\ldots+x_n$ in a reset phase. This lays a foundation for reduction of analysis of sequential NCL gates behavior to the check of corresponding combinational networks for set and reset phases.

Figure 5. Removal of registers from NCL pipeline

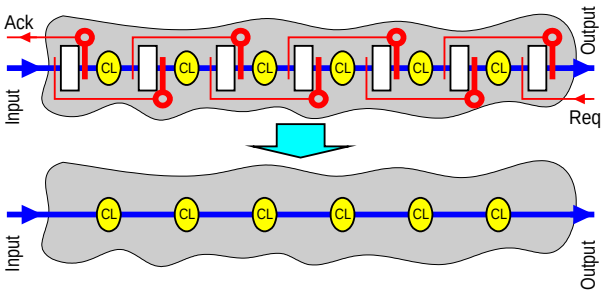

Figure 6 shows a NCL implementation C_{NCL} of a dual-rail MUX that selects one of the two data flows *a.0,a.1* and *b.0,b1* according to the value of control signals *s.0,s.1* (Figure 6 (a)). The behavior of the MUX in set and reset phases is captured by its Boolean "images" C_{Bool_set} (Figure 6(b)) and C_{Bool_reset} (Figure 6 (c)).

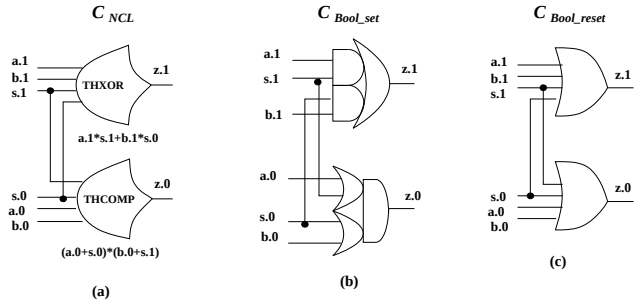

Figure 6. Reduction of a NCL network to Boolean networks

Proposition 2. An irredundant combinational circuit C_{NCL} of NCL gates is fully testable for all stuck-at faults.

The proof of Proposition 2 immediately follows from:

1. A set of stuck-at 0 faults of C_{NCL} being the same as a set of stuck-at 0 faults of its Boolean image C_{Bool_set} for a set phase.
2. A set of stuck-at 1 faults of C_{NCL} being the same as a set of stuck-at 1 faults of its Boolean image C_{Bool_reset} for a reset phase.
3. Full testability of irredundant networks from unate Boolean gates with respect to stuck-at faults [27]

According to Proposition 2, a test pattern generation for a NCL circuit could be solved by the generation of tests for

two Boolean networks (C_{Bool_set} and C_{Bool_reset}). Further simplification of this task is possible based on establishing a correlation between test patterns for stuck-at-0 and stuck-at-1 faults.

Proposition 3. Let a vector D_k from a valid DATA set be a test for a stuck-at 0 fault at input i of a NCL gate g. Then the test pair (D_k,*NULL*) tests a stuck-at 1 fault at input i of gate g.

Proof. D_k is a test for a stuck-at 0 fault at input i of gate g if it propagates a rising transition through some path $p = a,\ldots,i,g,\ldots,y$ from a primary input a to a primary output y. All nodes of p settle to a value "1" at the end of a set phase. In a fault-free circuit the following application of NULL at primary inputs results in a propagation of a falling transition through the path p that is observed at primary output y. If a node i of path p is stuck-at 1, then gate g cannot switch to "0" because its reset function is simply an OR of g fan-ins. From the similar consideration it follows that any node of path p in a transitive fanout of g (including primary output y) will also keep the value "1" and the stuck-at 1 fault is observable at y. ♦

Proposition 3 shows that analysis of a Boolean network C_{Bool_reset} is redundant during test pattern generation because if $T0=\{D_1,\ldots,D_m\}$ is a set of test patterns detecting stuck-at-0 faults then stuck-at-1 faults are automatically detected by applying test patterns $T=\{(D_1,NULL),\ldots,(D_m,NULL)\}$.

The above considerations result in the following test scenario for acyclic NCL pipelines:

1. Convert a pipeline into a NCL combinational circuit C_{ncl} by removing register circuitry.
2. Convert C_{ncl} into a Boolean combinational circuit C_{Bool_set} by replacing every NCL gate with the corresponding Boolean gate implementing the equivalent function.
3. Derive a set of test vectors $D_1,\ldots, D_m$ for checking stuck-at-0 faults in C_{bool_set} (to guarantee that $D_1,\ldots, D_k$ are valid DATA patterns, a constrained ATPG respecting one-hot encoding must be used).
4. Test the original pipeline by a testing sequence $(D_1,NULL),\ldots,(D_m,NULL)$

The suggested testing procedure has an additional advantage. If in a pipline with M stages a stuck-at fault occurs at stage i, then a completion detector of stage i does not produce a proper acknowledgment backward to stage i-1. Therefore new waverfronts could not propagate through i-1 stage and finally the pipeline stalls. The fact of stalling the pipeline is observed by a stuck-at of *Ack* line. Hence a comparison of responses of fault-free and faulty circuits at primary outputs is not needed for NCL pipelines because their faults are always observable at a single line (*Ack*). Note however, that to propagate the information about stalling the pipeline from the last stage to the first one, the last pair of testing patterns $(D_m,NULL)$ might need to be applied $M/2+1$ times (see the proof of Proposition 1 for explanation).

Finally we arrive to the following conclusion: **irredundant acyclic NCL pipelines are 100% stuck-at faults testable**.

Note, that this conclusion is stronger than the known result on stuck-at testability in arbitrary delay-insensitive circuits [28]. This is because in an NCL circuit the premature firings coming from stuck-at faults and occurring in one phase necessarily show up as a stall in the next phase of operation [1].

3.3 Cyclic Pipelines

In general, NCL circuits contain computational loops and are described by pipelines with one or more feedback paths. Figure 7 shows an example of a cyclic pipeline, which is the NCL equivalent of a typical accumulator circuit where the sum is fed back as one of the arguments.

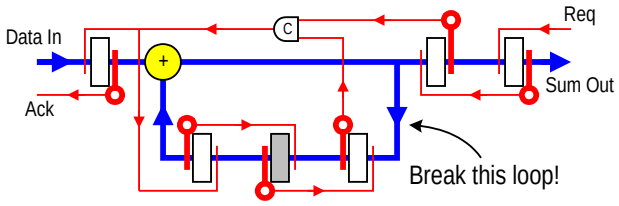

Figure 7. Cyclic pipeline with a computational loop

To employ the testing methods from Section 3.1 for cyclic pipelines, one needs to convert them into acyclic pipeline components. This can be done by applying a partial scan technique to break data-path loops.

3.3.1 Partial scan

The purpose of inserting scan cells in NCL pipelines is two-fold: a) to break the computational loops and b) insert test vectors into the data paths. The correctness of the pipeline under test is checked through observing the consistent changes of the *Ack* signals coming to the scan cells and acknowledging a propagation of wavefronts through the pipeline. This is done in the same way as discussed for acyclic pipelines.

[1] Unfortunately this is not true for other types of faults: stuck-at faults in internal feedbacks e.g. result in premature firings which do not lead to stalls of a pipeline.

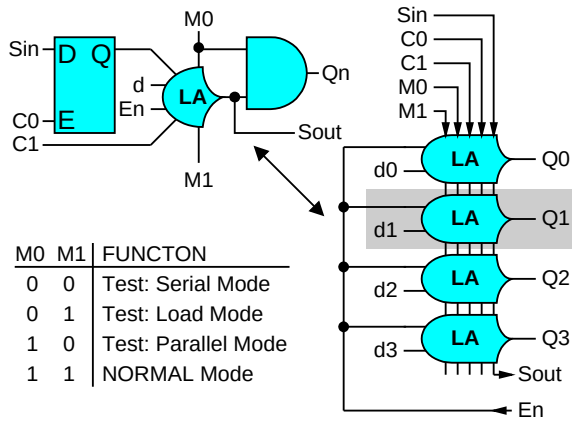

Figure 8. Cycle Scan cell model

M0	M1	Sin	C0	C1	d	En	Qn	Sout
0	0	0	↑↓	↑↓	x	1	0	0
		1	↑↓	↑↓	x	1	0	1
		x	x	x	x	0	0	0
0	1	x	0	↑↓	0	1	0	0
		x	0	↑↓	1	1	0	1
		x	x	x	x	0	0	0
1	0	x	0	0	x	1	Q_{LA}	Q_{LA}
		x	0	0	x	0	0	0
1	1	x	0	0	0	0	0	0
		x	0	0	0	1	Q_{LA}	Q_{LA}
		x	0	0	1	0	Q_{LA}	Q_{LA}
		x	0	0	1	1	1	1

Table 2. Scan cell truth table

Figure 8 illustrates an implementation of a scan register through the example of a 4-rail register with four scan cells. It targets LSSD style clocking [29] (2 phased non-overlapping clocks) for its single serial line. Table 2 defines the logic functionality of the scan cell, where *M0* and *M1* define the scan mode, *C0* and *C1* are non-overlapping clocks, *Sin* and *Sout* are scan input and output, *d* is a data input used in normal functioning and *En* is an enabling signal that is tied to *Ack* of the next registration stage.

The cell contains an input D-latch, an internal transparent latch (based on a C-element with additional control signals for scan modes) and the output AND gate (see Figure 8). Its static implementation requires 44 CMOS transistors.

Although it is not necessary to capture the resulting vectors from the stimuli, the scan cell provides that option with the *Load Mode* (*M0=0, M1=1*). The latter might be useful for fault localization. The scan register serves both as the source of scan vectors at the front of the pipeline as well as the detector of the proper changes of *Ack* signals.

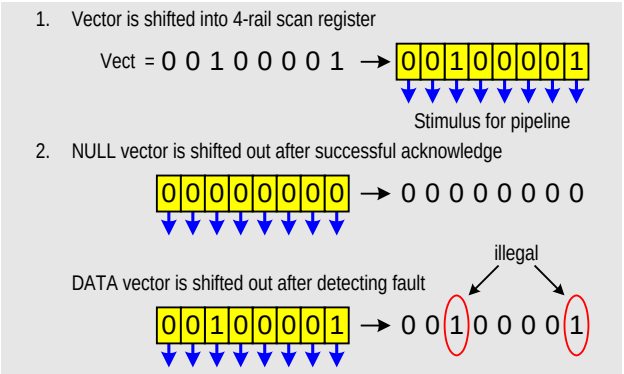

Figure 9. Scan protocol

In Figure 9, the scan protocol is illustrated graphically. Item 1 shows an example of a DATA test vector with the value '00100001' that is shifted into the scan register. Item 2 shows two possible results after the vector has been allowed to propagate down the pipeline. If the acknowledge signal is received from the downstream "registration" stage, then the contents of the scan register will be cleared to NULL. Subsequently, a NULL value will be shifted out when the new DATA vector is shifted in. On the other hand, if a fault has occurred and the pipeline is stalled, the acknowledge signal will not be received from the downstream "registration" stage, and the contents of the scan register will keep DATA. In this case, the corresponding DATA pattern will be observed shifting out of the scan register, indicating that a fault has occurred.

The following steps outline the test procedure using scan registers. Initially, *RESET* is asserted and the scan chain is set to **Serial Mode** (*M0=0, M1=0*).

1. Release RESET. The *Q* outputs of the scan chains stay NULL due to the *AND* gate at the output. In **Serial Mode** the 'd*n*' inputs are isolated from the C-latch.
2. While in **Serial Mode** (*M0=0, M1=0*) load the scan chain with the desired vector with the *SIN* pin using non-overlapping clocks *C0* and *C1*.
3. Set *M0 = 1* (*M1* remains at 0) to switch the scan chain to **Parallel Mode**. In this mode the acknowledge signal from the next downstream stage (*En* for a scan register) is at "1" (unless the pipeline under test is stalled), and data from the scan cells are passed to the data bus. At this time 'd*n*' inputs are blocked from loading into the scan chain register (see Table 2).
4. Wait for a predetermined amount of time to allow the slowest propagation of data through a single pipeline stage. If no faults have occurred the acknowledge signal on *En* goes to "0" and resets the transparent latch of the scan cell (scan register is reset to NULL).

The corresponding NULL wavefront propagates through the next stage of the pipeline, and sets *En* to "1" (unless the pipeline is stalled).

5. After the wait period, *M0* is reset to 0 (*M1* remains at 0). This forces the scan chain to switch into **Serial Mode** and produces NULL at all the scan chain outputs. The procedure goes back to step 3 and iterates.

When the system deadlocks because of a fault, the acknowledge signal that is fed to the scan register (input *En* of the scan cells) will be stuck either at DATA or at NULL. The latter can be detected by shifting a single DATA bit through the complete scan chain and analyzing *Sout* pin. One simply adds this extra DATA bit to the front of each scan vector.

At step 5 in the above procedure it is possible to load the contents of the last registration stage into the scan register before serial shifting the data out. This is useful for observing the resulting vectors after propagation through the pipeline. Load is enabled by a short positive pulse applied to *M1* while in the **Serial Mode**.

After all the vectors have been processed, it is necessary to perform extra handshakes to flush the pipelines (see the discussion on Proposition 1). The number of handshakes is defined by the length of the longest pipeline.

4. Experimental Results

The approach from Section 3.1 for test pattern generation in NCL combinational circuits was checked on more than 20 examples from the NCL regression suite. The examples include encoders, decoders, simple control structures, adders, shifters, etc. These circuits were constructed with the NCL™ Design Flow tool from Theseus Logic, Inc. The TetraMAX ATPG tool from Synopsys was used to generate test vectors for stuck-at faults. For all examples but one (encoder16to4) TetraMax showed a 100% test coverage. Lower than 100% test coverage (96%) for the encoder example stems from the redundancy of the implementation obtained by NCL automatic design flow. Note that this problem is not specific to NCL flow because the very same specification pushed through Design Compiler from Synopsys produces a redundant synchronous circuit due to poor optimization for hierarchical specifications. The redundancy is eliminated when using flattened (rather than hierarchical) encoder specification. The latter showed 100% test coverage as expected from theory.

	NCL8051 ALU	VITERBI DCDR	SINGLE DES
NCL gate count	1188	2290	3615
Transistor count w/o scan	22284	50632	65448
Transistor count with scan	24324	62272	72888
Length of scan chain	68	388	240
Number of I/O pins	177	24	396
Number of Test Vectors	120	429	348+369
Area penalty due to scan	9.2%	23.0%	11.4%

Table 3. Experimental results

Three major NCL designs, the Viterbi Decoder, the Single DES, and the NCL8051 ALU, were used as test cases to verify the test methodology using scan. In all cases, the use of partial scan together with TetraMax ATPG tool resulted with near 100% stuck-at fault coverage. Table 3 gives a summary of the results.

In the experiments, the points of partial scan insertion were defined manually by a designer. In the future this task can be automated using Theseus Logic's *NCL shell* CAD tool. *NCL shell* analyzes the structure of cycles in a design, thus insertion of scan is possible following the min-cut algorithm e.g.

The ATPG setting for TetraMax was done exactly in the way presented in Section 3.1. The test patterns generated by TetraMax are applied to the NCL circuit using the ModelSim simulator from Mentor Graphics. A fault grading tool confirmed 100% stuck-at fault coverage. The only exception was the Single DES design which had a 98.4% coverage – a result that would most likely go to 100% with refinement of ATPG settings.

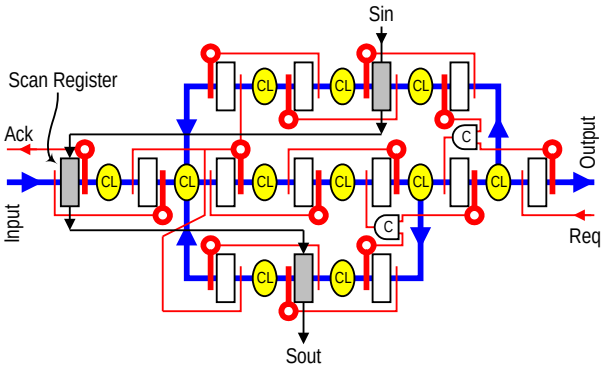

Figure 10. NCL system with Scan insertion

During ATPG the complicated functional model of the NCL Cycle Scan register is replaced in TetraMax by a clock primitive scan cell flip-flop without influencing the test coverage.

Note that partial scan insertion does not require balanced pipeline lengths. For example, Figure 10 shows loops of varying sizes, resulting in different linear pipeline lengths when they are broken. It follows that as the lengths of the pipelines increase, the area impact of scan insertion will decrease. This is because only a single scan register is required per acyclic pipeline.

The minimum length allowed for an NCL data loop is 3 "registration" stages. The Viterbi Decoder circuit exhibits the worst case scan overhead because all its loops are shallow (see Figure 11 where scan registers are shown by shadowing).

The *Load Mode* for the scan register was simulated for the Viterbi Decoder circuit, and the shift-out vectors matched perfectly the *unload* vectors generated by the TetraMAX ATPG tool.

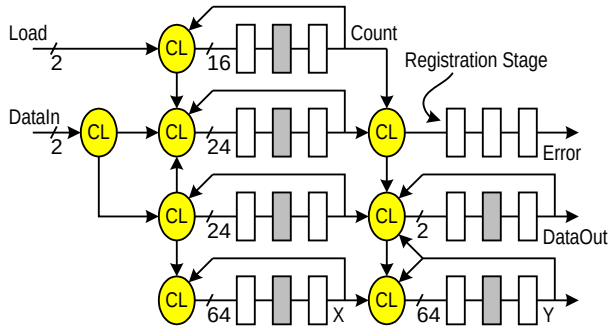

Figure 11. Viterbi Decoder architecture

5. Conclusion

The paper presents the new methodology for testing NCL circuits. The major advantages of the methodology are:

1. It is highly effective (allows a designer to achieve 100% stuck-at fault coverage)
2. Its overhead is relatively low because of the use of partial scan techniques rather than full scan.
3. It targets the use of commercial testing CAD tools available in today's market.

In this way the methodology benefits from both worlds: the known simplicity of stuck-at fault testing in asynchronous circuits with the ease of testing procedures from synchronous design.

Future improvements are possible that will:

1. Automate the scan insertion and reduce the scan overhead
2. Extend the approach beyond the input stuck-at fault model (testing the gate internal feedbacks e.g.)
3. Use of other than partial scan techniques (combine functional testing of microprocessors with the ATPG techniques for choosing the data content of instructions e.g.)

6. Acknowledgments

The authors wish to thank Ad Peters who attracted our attention to the importance of testing the internal feedbacks in NCL gates and Bob Duell and Ritesh Banglani for their valuable help in setting ATPG flow for NCL circuits.

7. References

1. Shai Rotem, Ken Stevens, Ran Ginosar, Peter Beerel, Chris Myers, Kenneth Yun, Rakefet Kol, Charles Dike, Marly Roncken, and Boris Agapiev. RAPPID: An asynchronous instruction length decoder. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 60-70, April 1999.
2. Ivan Sutherland and Jon Lexau. Designing fast asynchronous circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 184-193. IEEE Computer Society Press, March 2001.
3. Kees van Berkel, Ronan Burgess, Joep Kessels, Ad Peeters, Marly Roncken, and Frits Schalij. A fully-asynchronous low-power error corrector for the DCC player. In *International Solid State Circuits Conference*, pages 88-89, February 1994.
4. S. B. Furber, D. A. Edwards, and J. D. Garside. AMULET3: a 100 MIPS asynchronous embedded processor. In *Proc. International Conf. Computer Design (ICCD)*, September 2000.
5. W.A. Lien, P.Day, C.Faransworth, D.L.Jackson, J.Liu, and N.Paver. Noise in a self-timed and synchronous implementation of a DSP. Unpublished, White Paper, Cogency Technology Inc., 1997.
6. John McCardle, David Chester. Measuring an Asynchronous Processor's Power and Noise, SNUG Conference, April 2001.
7. K.Y. Yun, D.L. Dill, and S.M. Nowick. Synthesis of 3D asynchronous state machines. In Proc. International Conf. Computer Design (ICCD), pages 346-350. IEEE Computer Society Press, October 1992.
8. Steven M. Nowick and David L. Dill. Automatic synthesis of locally clocked asynchronous state machine. In Proc. International Conf. Computer-Aided Design (ICCAD). IEEE Computer Society Press, pp.318-321, November 1991.
9. Alain J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, Developments in Concurrency and Communication, UT Year of Programming Series pages 1-64. Addison-Wesley, 1990
10. K. van Berkel. Handshake Circuits: an Asynchronous Architecture for VLSI Programming, Cambridge University Press, 1993.
11. Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT Laboratory for Computer Science, June 1987.

12. V.I. Varshavsky, editor. Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.
13. M. Renaudin, P. Vivet, and F. Robin. A design framework for asynchronous/synchronous circuits based on CHP to HDL translation. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 135-144, April 1999.
14. Ivan Blunno and Luciano Lavagno. Automated synthesis of micro-pipelines from behavioral Verilog HDL. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 84-92. IEEE Computer Society Press, April 2000.
15. Michiel Ligthart, Karl Fant, Ross Smith, Alexander Taubin, and Alex Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 114-125. IEEE Computer Society Press, April 2000.
16. K. Fant and S.A. Brandt. NULL conventional logic: A complete and consistent logic for asynchronous digital circuit synthesis. In International Conference on Application-specific Systems, Architectures, and Processors, pages 261-273, 1996.
17. Peter Beerel and Teresa Meng. Semi-modularity and self-diagnostic asynchronous control circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI*, pages 103-117. MIT Press, March 1991.
18. Oriol Roig, Jordi Cortadella, Marco A. Pe na, and Enric Pastor. Automatic generation of synchronous test patterns for asynchronous circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 620-625, June 1997.
19. Ajay Khoche and Erik Brunvand. Testing self-timed circuits using scan paths. In *5th NASA Symposium on VLSI Design*, November 1993.
20. O. A. Petlin and S. B. Furber. Scan testing of micropipelines. In *Proc. IEEE VLSI Test Symposium*, pages 296-301, May 1995.
21. Marly Roncken. Partial scan test for asynchronous circuits illustrated on a DCC error corrector. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 247-256, November 1994.
22. Marly Roncken. Defect-oriented testability for asynchronous IC's. *Proceedings of the IEEE*, 87(2):363-375, February 1999.
23. W. J. Bainbridge and S. B. Furber. Delay insensitive system-on-chip interconnect using 1-of-4 data encoding. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 118-126. IEEE Computer Society Press, March 2001.
24. K. Fant, R. Stephani, R. Smith, R. Jorgenson. The orphans in 2 value NULL convention logic. Theseus Logic, 1998, (http://www.theseus.com/PDF/orph_paper.pdf)
25 Alexander Saldanha, Robert Brayton, and Alberto Sangiovanni-Vincentelli. Equivalence of robust delay-fault and single stuck-fault test generation In Proc. ACM/IEEE Design Automation Conference, pages 173--176,1992.
26. M. Abramovici, M. Breuer, A. Friedman. Digital Systems Testing and Testable Design, Computer Science Press, 1990
27. D.Scherz, G. Metze. On the design of multiple fault diagnosable networks. IEEE Trans. Comput., vol. C-21, August, 1972
28. Pieter J. Hazewindus. *Testing Delay-Insensitive Circuits*. PhD thesis, California Institute of Technology, 1992.
29. E.B.Eichelberger and T.W.Williams. A logic design structure for LSI testability. *Proc. ACM/IEEE Design Automation Conf.*, pages 462-486,1977