Design of Asynchronous Circuits Using Synchronous CAD Tools

Alex Kondratyev
Cadence Berkeley Laboratories

Kelvin Lwin
Reshape Inc.

Poor CAD support hinders wide acceptance of asynchronous methodologies, and asynchronous design tools are far behind synchronous commercial tools. A new design flow, NCL_X, based entirely on commercial CAD tools, targets a subclass of asynchronous circuits called null convention logic. NCL_X shows significant area improvement over other flows for this subclass.

EDA FLOWS are industry driven, and thus use synchronous methodologies as de facto standards. However, implementation problems arise from imposing a synchronous model of operation on deep-submicron circuits. This problem motivates the investigation of other, asynchronous modes of operation. Acceptance of new design methodologies, including asynchronous ones, by engineering and industrial communities depends on three major issues:

- added value of the methodology in terms of area, power, speed, electromagnetic interference (EMI), noise immunity, and so on;
- tradeoffs, which design parameters often worsen to achieve added values, such as speed versus power, and area versus EMI; and
- the cost of switching to the new methodology, including training time, development of new libraries, and time spent dealing with CAD tool immaturity.

Researchers have demonstrated that asynchronous designs can deliver higher speeds because they can latch values as the computation finishes, unlike synchronous design, which must wait for all computations to finish before latching. This improves overall performance when the average case finishes much earlier than the worst case.[1] Researchers have also shown that, because of the absence of a clock and natural support of idle mode, such designs might consume less power than synchronous designs.[2,3] Finally, asynchronous designs incur low EMI and noise, thanks to their even time distribution of switching activities.[4] (See the "Practical asynchronous circuits and tools" sidebar for another perspective on asynchronous design from a leading expert in the field.)

Nevertheless, industry acceptance of asynchronous designs has been slow because most success stories have thus far not delivered everything they've promised. Asynchronous high-speed circuits are custom designs,[1] incorporating complicated timing assumptions about circuit delays and thus blurring the boundary between asynchronous and synchronous styles. CAD support for such methodologies is even more problematic than for custom synchronous designs.

Thus, low-power asynchronous circuits usually compromise by using asynchronous meth-

0740-7475/02/$17.00 © 2002 IEEE

Practical asynchronous circuits and tools

Alain J. Martin, California Institute of Technology

After looking like a pipe dream for many years, asynchronous technology is becoming a viable—and perhaps unavoidable—alternative to clocked design for large VLSI systems. Asynchronous technology rests on local communications among concurrent units. Handshake protocols implement communication and synchronization among those units. There is no concept of global time—no clocks—and no assumptions about the duration of an action or communication. Asynchronous circuits have several advantages:

- They avoid all issues related to distributing a clock signal reliably and efficiently across a large chip.
- Because they can be largely insensitive to delay variations, asynchronous circuits can tolerate large variations in a design's physical parameters, which are difficult to control in deep-submicron technology.
- They offer, to the designer of low-power systems, automatic and perfect shut-off of idle parts.
- Asynchronous technology lends itself to high-level synthesis and modular design.

The asynchronous community has made spectacular progress in the past decade, and today we know how to design correct and efficient asynchronous circuits. The correctness issue mostly concerned designing glitch-free circuits. The efficiency issue related to the cost of handshake protocols and completion detection.

To appreciate this progress, consider the family of asynchronous chips designed at the California Institute of Technology between 1989 and 1999. Researchers at Caltech designed the world's first asynchronous microprocessor in 1989. The chip had 20,000 transistors and was a simple 16-bit machine. Its peak performance was 5 MIPS at 2 V drawing 10 mW, and 18 MIPS at 5 V drawing 225 mW, in 1.6-micron CMOS. It was correct on first silicon, and its performance was competitive with designs of that time.

In 1994, Caltech presented an asynchronous, pipelined lattice structure filter, the first example of very fine pipelining. The chip had 250,000 transistors. In 0.9-micron CMOS and at 3.3 V, the throughput was 130 MHz—that is, 500 million 12-bit additions or multiplications per second. In liquid nitrogen, the filter executed 1 billion operations per second. The chip worked correctly from 1 V to 5 V. At 1.1 V, it operated at 36 million operations per second and consumed 20 mW.

Between 1995 and 1998, Caltech researchers designed the MiniMIPS, an asynchronous MIPS R3000. As of today, it is still the most efficient asynchronous chip ever designed. The R3000 is a classic 32-bit RISC processor with two 4-Kbyte caches. Caltech fabricated the chip in 0.6-micron CMOS. The transistor count was 2 million. All chips were functional, except one with a defective package. The test performance on small programs was 180 MIPS and 4 W at 3.3 V, 100 MIPS and 850 mW at 2.0 V, and 60 MIPS and 220 mW at 1.5 V. The performance figures running Dhrystone benchmarks were 185 MHz at 3.3 V (165 VAX MIPS).

So, if researchers have resolved most of the issues regarding the design of asynchronous circuits, why is the industry so slow in adopting the technology? There are certainly sociological and other nontechnical answers to the question. But Kondratyev and Lwin are correct in identifying the absence of design tools as the single most important technical stumbling block. As long as there is no market for asynchronous EDA tools, the EDA industry will not create these tools. But as long as there are no tools for asynchronous design, there will be no market for the tools! Kondratyev and Lwin aim to break free from this vicious cycle by adapting existing tools to the asynchronous design flow. Whether they can accomplish this without too high a performance penalty remains to be seen. But the experiment is worth watching.

Alain J. Martin is a professor of computer science at the California Institute of Technology. Contact him at alain@async.caltech.edu.

ods for control synthesis and synchronous methods for data path design. Communication among these circuit elements rests on timing assumptions and delay-matching mechanisms. The latter makes verifying such circuits difficult.

To further complicate the situation, a lack of commercial CAD support for asynchronous synthesis sometimes forces designers to use in-house specification languages and design tools.[2,3] This chronic deficiency is a major roadblock to wider acceptance of asynchronous methodologies.

Low EMI and noise coefficients are the only "free" advantages of asynchronous circuits.

Without the clock, noise and EMI spectrums are significantly flatter across the entire frequency domain. For noise, this can be a 10-dB drop, according to McCardle and Chester.[4]

Until recently, EMI and noise metrics were second-class citizens; everybody focused on power and performance. But EMI and noise metrics are garnering more attention because of two emerging applications: mixed-signal design and smart cards. In the former, analog functions are particularly sensitive to clock-correlated, digital-switching noise. Reducing noise and EMI significantly boosts both precision and performance. In smart cards, EMI doesn't affect functionality but has a significant impact on security. Noninvasive security attacks depend on monitoring a smart card's *power rail*, or EMI signature, to decipher information on the card. Even distribution of circuit-switching activities vastly improves security.

We propose an automatic design flow for asynchronous circuits, with the following features:

- Its added value rests on its low EMI and/or higher security level.
- This value comes at the expense of an area penalty.
- The flow has low switching costs because it closely mimics the conventional synchronous hardware description language (HDL) methodology and relies on commercial design tools.

The last feature is key for asynchronous design. It removes the roadblock of reeducating designers and shifts the criteria for choosing whether to use asynchronous circuits toward an objective estimation of their tradeoffs: area, speed, power, and so on. This new design flow, NCL_X, targets a subclass of asynchronous circuits called null convention logic (NCL).

Asynchronous design styles

Clocking is a common, simple abstraction for representing the timing issues in the behavior of real circuits. Generally speaking, it lets designers ignore timing when considering system functions. Designers can describe both the functions performed and the circuits themselves in terms of logical equations (Boolean algebra). In general, synchronous designers don't need to worry about the exact sequence of gate switching as long as the outputs are correct at the clock pulses.

In contrast, asynchronous circuits must strictly coordinate their behavior. Logic synthesis for asynchronous circuits not only must handle circuit functionality but also must properly order gate activity (switching). The solution is to use functional redundancy to explicitly model computation flows without using abstract means such as clocks. Using logic to ensure correct circuit behavior under any delay distribution can be costly and impractical. Therefore, most asynchronous design styles use certain timing assumptions to correctly align functions.

These assumptions can have different degrees of locality—from matching delays on some wire forks to balancing all system paths (as in synchronous methodologies). Localized assumptions are easier to meet in a design because they simplify timing convergence problems and provide more modularity. But ensuring the correctness of such assumptions can be costly because it requires more system redundancy at the functional level. Asynchronous design styles differ in the way they handle the tradeoff between locality of timing assumptions and design cost.

Existing asynchronous design flows include the following:

- *Delay-insensitive (DI)* circuits impose no timing assumptions, allowing arbitrary gate and wire delays.[5] Unfortunately, the class of DI implementations is limited and impractical.
- *Quasi-delay-insensitive (QDI)* circuits partition wires into critical and noncritical categories.[6] Designers of such circuits consider forks in critical wires to be safe by assuming that the skew caused by their wire delays is less than the minimum gate delay. Designers thus assume these wires to be isochronic. In contrast, noncritical wires can have arbitrary delays.
- *Speed-independent (SI)* circuits let gates have any length of delay, but wire delays must be negligible.[5]

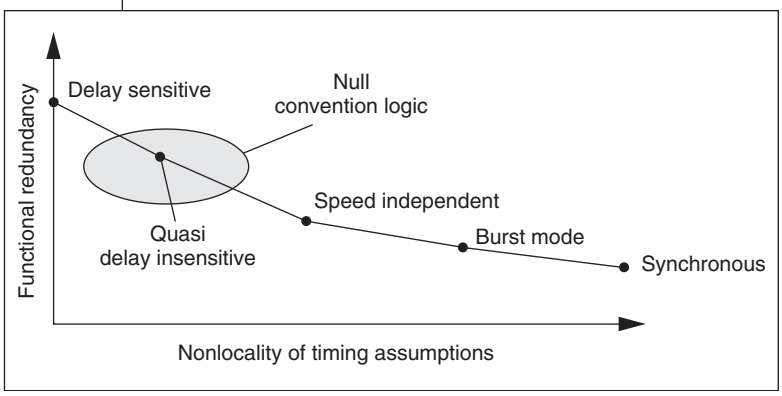


Figure 1. Functional redundancy and the nonlocality of timing assumptions for different asynchronous design flows. The gray area indicates that NCL has more assumptions than in typical quasi-delay-insensitive circuits but that each assumption is safer.

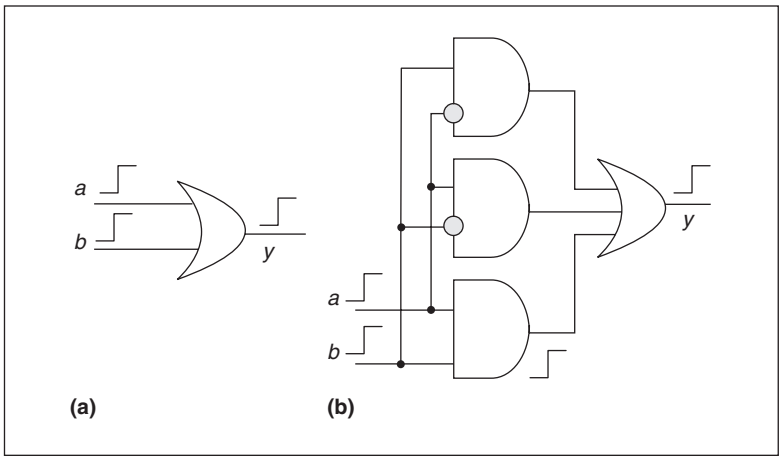


Figure 2. Illustration of acknowledgment by outputs. The transition at *y* does not acknowledge the input transitions at *a* and *b* (a); output *y* properly acknowledges the same input transitions (b).

- *Burst-mode (BM)* circuits rely on the fundamental mode protocol, which applies a new input pattern to the circuit only when the circuit has completely settled to its steady state after the previous pattern.[7]

As Figure 1 shows, the locality of timing assumptions decreases, from DI systems (which incorporate no assumptions) to burst-mode circuits (in which timing assumptions involve global characteristics of system behavior, and the environment's response is slower than the module's delay).

Our work targets NCL,[8,9] which fits the QDI methodology because it imposes timing assumptions only on wire forks. However, rather than assuming isochronic forks, NCL requires the skew after the fork to be less than the circuit response time. This change makes it far easier for the design to satisfy timing constraints.

However, NCL doesn't distinguish between critical and noncritical wires, and thus designers must ensure that *all* wire forks meet timing assumptions. Although this change requires more work, it relieves designers from having to determine whether a fork is critical or not. In this way, the NCL design flow is more likely to produce an acceptable design, but it must check more timing assumptions than a typical QDI design flow.

Delay-insensitive combinational circuits

A combinational gate output reflects the gate's Boolean function after its gate delay. Certain functions let you infer the input state by observing the output. For example, output 0 for a two-input OR gate means both inputs are 0. For the other three possible inputs, the output is 1. When transitioning from one set of inputs to another, the output can temporarily become 0 before returning to its proper value of 1. This behavior constitutes a *hazard*. A circuit in which hazards cannot occur under any distribution of gate and wire delays is delay insensitive.

Imagine a circuit in which false transitions don't occur and where you can always infer the inputs by merely observing the outputs. You could say that the output of such a circuit has full *acknowledgment* of all its inputs. The notion of acknowledgment is key to ensuring delay insensitivity. If every transition at a wire or gate in a circuit translates its firing results into changes in the primary outputs, the circuit behavior does not depend on transition timing and is delay insensitive.

Figure 2 illustrates the acknowledgment concept. In Figure 2a, the transition at *y* does not properly acknowledge the rising input transitions at both *a* and *b*, because *y* changes as soon as any one of its inputs transitions, regardless of the value at the other input. Output *y* properly acknowledges the same input transitions in the circuit in Figure 2b because, in this

case, y does not change until the circuit asserts both a and b as 1.

Null convention logic

NCL is a specific way of implementing data communication based on DI encoding. Data changes from the spacer (Null) to a proper code word (Data) in the set phase, and then back to Null in the reset phase. NCL targets simple DI encoding in which Data code words are *one-hot codes* (only 1 bit of the code can be asserted to 1), and a vector with all entries equal to 0 represents the spacer Null. For example, in dual-rail encoding, two wires, $a.0$ and $a.1$, represent each signal, a. Thus, this method encodes $a = 1$ as $a.0 = 0$ and $a.1 = 1$, and $a = 0$ as $a.0 = 1$ and $a.1 = 0$. DI encoding lets the receiver determine that a code word has arrived by observing the code word itself, without appealing to timing assumptions. In particular, for dual-rail signals $a.0$ and $a.1$, an OR gate ($a.0 + a.1$) is the simplest detector for validating a code word at $a.0$ and $a.1$.

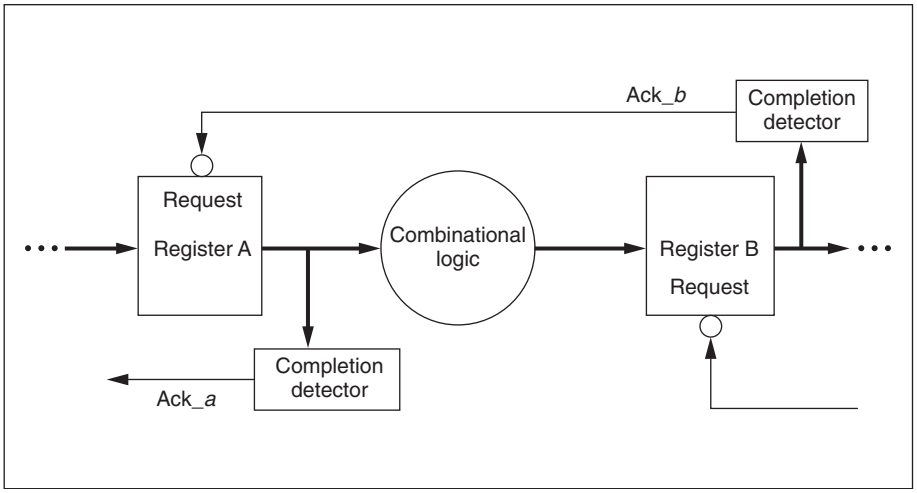


Figure 3. NCL system implementation.

At an architectural level, NCL systems clearly separate sequential and combinational parts, much in the same way as with synchronous systems, as Figure 3 shows.

NCL systems borrow the idea of organizing register interaction in DI fashion from micropipeline architectures.[10] The main difference concerns data path implementation: It is delay insensitive for NCL and synchronous for micropipelines.[11]

To understand how the NCL system functions, assume that all registers are initially in the Null state and that the circuit has asserted *Ack* signals to 0. When Data arrives, a register's outputs change from Null to Data, and the Data wave front propagates through a combinational circuit to the next register's inputs. Simultaneously, a completion detector checks for a Data code word at its inputs, and replies by raising the *Ack* signal. This signal disables the previous register's request line and prepares the register for storing the next Null wave front. The request-acknowledgment mechanism of register interaction ensures a two-phase discipline in NCL system functioning and prevents collisions between different Data wave fronts.[10]

Guaranteed implementation of this behavior requires gates, such as those based on NCL, that satisfy the following properties:

- Monotonic Null → Data (Data → Null) transitions at a combinational circuit's inputs result in monotonic Null → Data (Data → Null) transitions at its outputs. These properties are achievable by using gates that implement a *positively unate function* (all inputs in an inversionless function). Each gate can then make at most one transition in the set or reset phases, much like in precharged dynamic circuits.
- For intermediate states of the Null → Data input transition, a combinational circuit must keep some of its outputs in Null (so that it does not produce Data prematurely). For intermediate states of the Data → Null input transition, the circuit must keep some of its outputs in Data (so that it does not produce Null prematurely). Thus, NCL gates must track the current function; that is, gates must have internal memory.

These conditions lead to a general representation of an NCL gate as $g = S + gR'$, where S and R are the unate set and reset functions. Because there's only one designated value for Null (the vector with all 0s), the system must

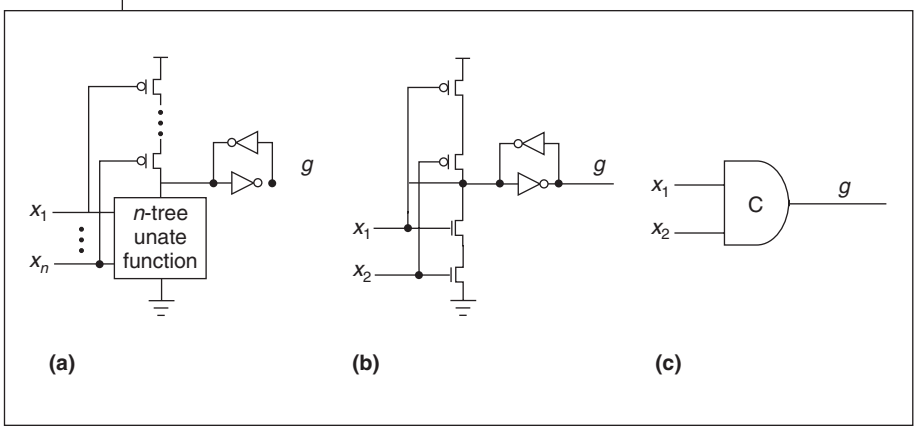


Figure 4. Semistatic CMOS NCL gate implementation (a), a particular NCL gate known as Muller's C-element (b), and its notation (c).

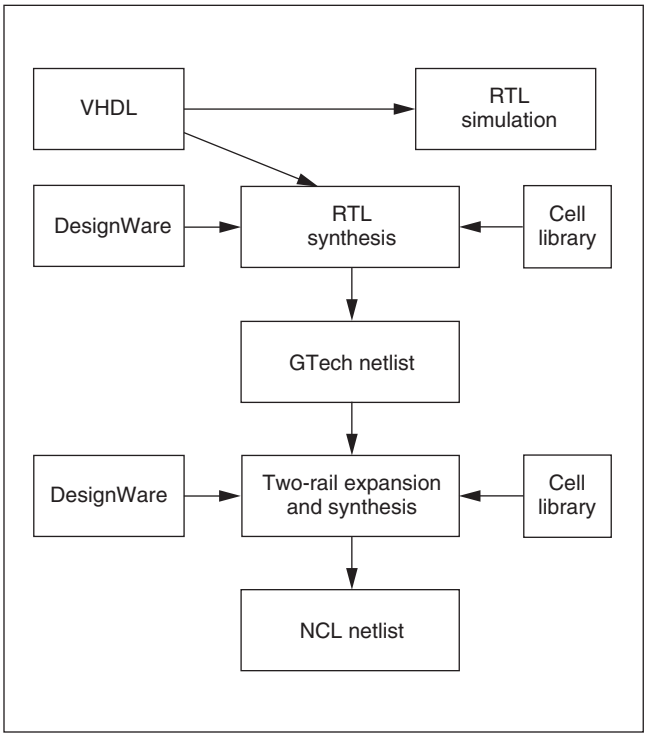


Figure 5. RTL flow for NCL.

uniformly reset every gate by changing its output to 0 only when all inputs to that gate are 0.

This refines the representation of gate g to the threshold function $g(x_1, x_2, \ldots, x_n) = S + g(x_1 + x_2 + \ldots + x_n)$. Figure 4a shows a semistatic CMOS implementation of an NCL gate. Figures 4b and 4c show an implementation and notation for a particular NCL gate with the function $g = x_1x_2 + g(x_1 + x_2)$, known from literature as a Muller's C-element.

NCL design flow

NCL is coded at the register-transfer level (RTL). To synthesize and simulate an NCL circuit at the RTL using commercial tools, the tools must handle the Null value and sequential behavior of threshold gates. Designers must

- separate combinational logic and registers, writing combinational logic as concurrent signal assignments or in processes; and
- instantiate NCL registers and provide a simulation-only model for sequential behavior of NCL gates but ignore the simulation model during synthesis. For synthesis, threshold gates are represented by their set functions and look like Boolean gates.

As Figure 5 shows, the NCL design flow uses off-the-shelf simulation and synthesis components.

The flow executes two synthesis steps. The first step treats NCL variables as single wires. The synthesis tool performs HDL optimizations and outputs a network built from components in the Synopsys GTech (generic technology) library, as if it were a conventional Boolean RTL circuit. The second step expands the intermediate GTech netlist into a dual-rail NCL by making dual-rail expansions and mapping them into the threshold library. The details of these two implementation steps can affect the quality of the final results.

Ligthart et al. suggested a regular method for NCL implementation,[8] which we call NCL_D, based on delay-insensitive minterm synthesis (DIMS).[1] This method implements these two steps as follows:

1. It maps the optimized network into two-input NAND, NOR, and XOR gates.
2. It first represents each wire as a dual-rail pair, $a.0$ and $a.1$, and then directly translates two-input Boolean gates into threshold gate pairs with limited optimization of a threshold network.

Figure 6 shows a step-2 implementation for two-input NAND gate $c = (a, b)$. (In the remaining figures, a "T" inside a gate symbol indicates a threshold gate; a "C" indicates a Muller's C-element.)

Therefore, for gates and wires before forks, this translation scheme supports circuits that are delay insensitive by construction; for wires after forks, justifying correctness requires a review of timing assumptions.

The main advantages of NCL_D circuits are the translation scheme's simplicity and the automatic verification of DI properties during implementation. Unfortunately, NCL_D circuits incur significant overhead, which comes from two main sources:

- overdesigning because of the locality of DI property verification (no sharing in the acknowledgment is allowed), and
- little room for optimization (optimization can easily destroy DI properties).

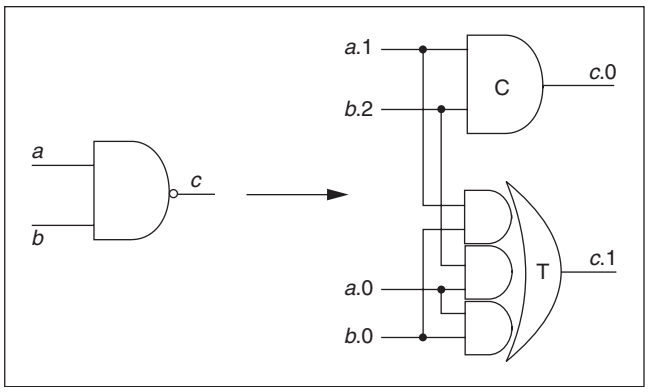


Figure 6. Dual-rail expansion for a NAND gate.

NCL flow with explicit completeness

Our design flow, NCL_X, exploits the idea of separate implementations for functionality and delay insensitivity. NCL_X partitions an NCL circuit into functional and completion parts, allowing independent optimization of each. Modifying the flow's implementation steps permits this separate implementation and optimization. In step 1, designers perform a conventional logic synthesis (with optimization) from the RTL specification of an NCL circuit. This step also maps the resulting network into a GTech library, using gates that implement set functions of threshold gates. Step 2 involves the following substeps:

- reducing the logic network to unate gates by using two different variables, $a.0$ and $a.1$, for direct and inverse values of signal a (the obtained unate network implements *rail*.1 of a dual-rail combinational circuit);
- enabling dual-rail expansion of the combinational logic by creating a corresponding dual gate in the *rail*.0 network for each gate in the *rail*.1 network; and
- ensuring delay insensitivity by providing local completion detectors (OR gates) for each pair of dual gates and connecting them in a completion network (multi-input C-element) with a single output, *done*.

Implementing the NCL_X design flow requires a minor modification of interfacing conventions within the NCL system. We assume that for each two-rail primary input ($a.0$, $a.1$), explicit signal *a.go* exists such that $a.0 \neq a.1 \rightarrow a.go = 1$ (set phase), whereas $a.0 = a.1 = 0 \rightarrow a.go = 0$ (reset phase). Figure 7 (next page) shows the modified organization of the NCL system. Unlike the system in Figure 3, this system has separate completion detectors for combinational logic and registers.

Designing the combinational logic for a 4-to-2 encoder illustrates this design flow. Figure 8 gives the RTL specification for this encoder.

Figure 9 (page 115) shows the design steps in the NCL_X implementation of the encoder. C-element *In.go* provides information about the validity of input code words. This element combines all completion signals for primary inputs. Output *I.go* connects to C-element *done*.

In an NCL_X implementation, the only unacknowledged transitions possible occur at wires after a fork. For these transitions, correct circuit behavior depends on the timing assumption that bounds the possible skew of wire delays after the fork for the duration of the set (reset) phase. This assumption is very conservative and easily testable. Apart from wire fork points, an NCL_X circuit is delay insensitive.

A clear boundary between a circuit's func-

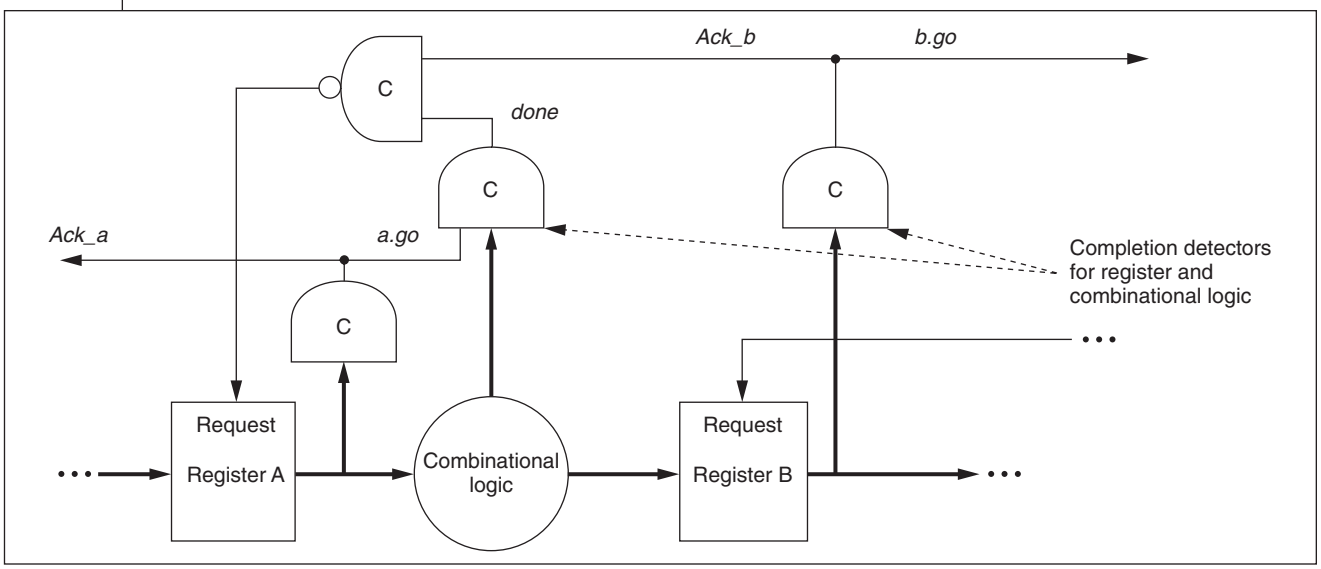


Figure 7. NCL_X produces an NCL system with explicit completion detection.

```
encode : process(din)
    begin
      if din = "1000" then
         d <= "11";
      elsif din = "0100" then
         d <= "10";
      elsif din = "0010" then
         d <= "01";
      elsif din= "0001" then
         d <= "00";
      else
         d <= (others => '0');
      end if;
```

Figure 8. RTL specification for a 4-to-2 encoder.

tional and completion parts significantly simplifies optimization procedures.

Completion part optimization

Some internal gates can receive an acknowledgment through the functional parts. In these cases, their local completion detectors are redundant, and thus removable without any influence on DI properties. Such an analysis can be automatic and help optimize the completion network. The 4-to-2 encoder example illustrates this specific type of optimization.

For the functional part, gate pairs ($x.0$, $x.1$) and ($y.0$, $y.1$) cannot switch until primary inputs *in*1, *in*2, and *in*3 receive the settled dual-rail values. Hence, ($x.0$, $x.1$) and ($y.0$, $y.1$) provide an acknowledgment for *in*1, *in*2, and *in*3; and signals *in*1.*go*, *in*2.*go*, and *in*3.*go* can be removed from input completion detector *In*.*go*. This in turn simplifies *In*.*go* to a single wire, providing significant area savings (about 35%) for the completion part.

Beyond QDI class

Accepting more stringent timing assumptions than QDI provides further opportunities. Many designs simplify asynchronous circuits by using timing assumptions.[1] The common bottleneck in such designs is in analysis and justification of the imposed assumptions. Fortunately, the strict separation between the set and reset phases in NCL system behavior is sufficient for verifying timing assumptions within the boundaries of a single phase. In this way, timing analysis for NCL systems strongly resembles static timing analysis for synchronous circuits.

If, for example, the *done* signal comes earlier than a register's acknowledgment signal, the entire completion network atop the functional part is redundant. This condition is easy to check through static timing analysis:

longest_functional_path + *CL_compl_detector* < *shortest_functional_path* + *register* + *reg_compl_detector*.

If this check fails, you could split the func-

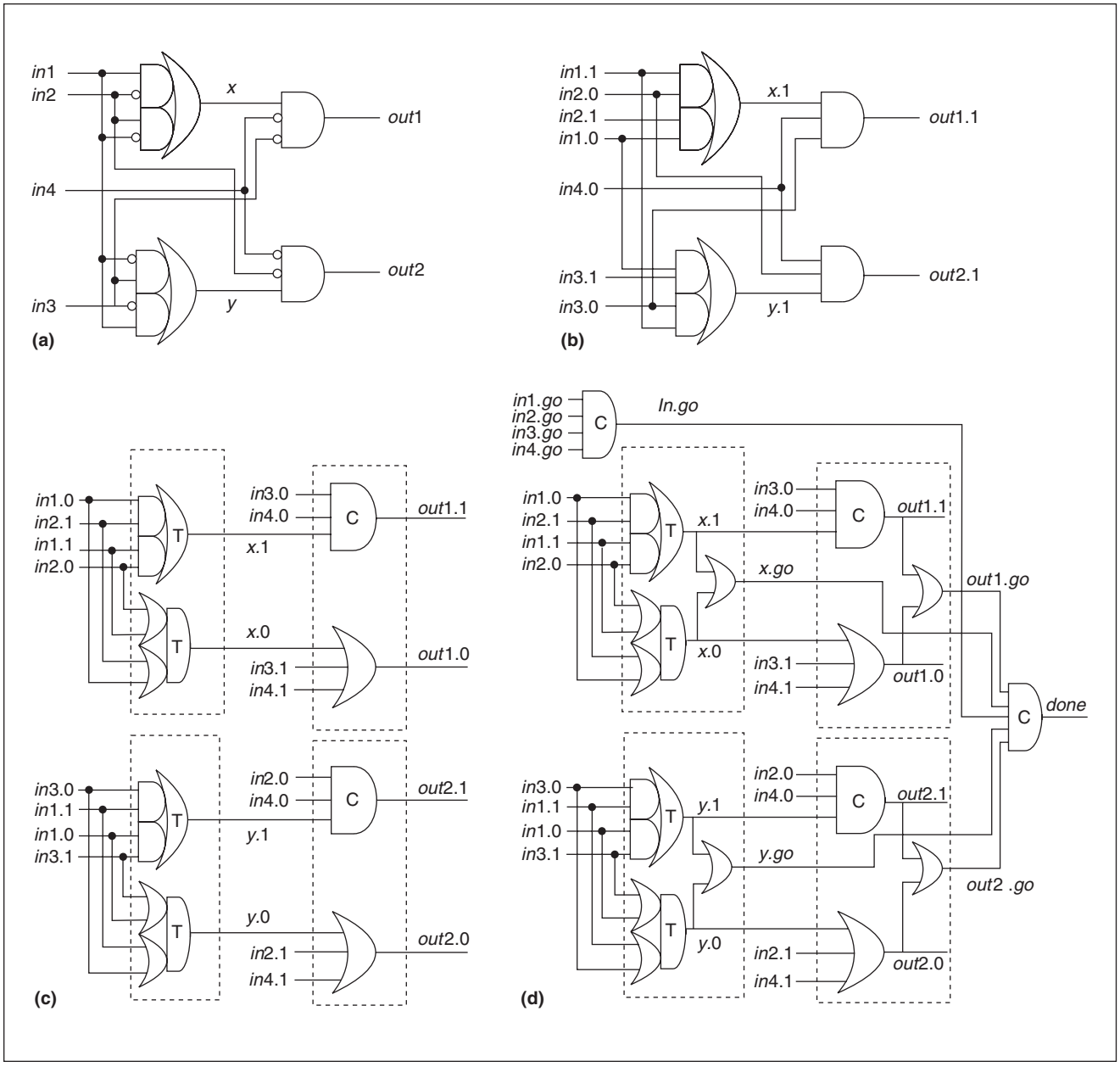


Figure 9. Encoder is first mapped into a GTech library (a). After the automatic tool reduces the logic network to unate gates and provides a dual function for each gate in unate representation (b), it can expand the network into a dual-rail implementation (c). Each gate pair contributes an output through a local completion detector (OR gate) that goes to a multi-input C-element, which provides the *done* output signal (d).

tional part's completion network into pieces and use the weaker timing conditions for the subnetworks.

Comparison of NCL_D and NCL_X

We compared NCL_X with Ligthart's NCL_D,[8] based on area and power consumption. Speed estimation was less of an issue because our design flow does not target high-speed applications. However, preliminary data shows that NCL_X circuits do not degrade performance any more than NCL_D circuits.

Area

We kept the front-end RTL coding style for the NCL_X the same as for the NCL_D. This let us synthesize the same designs for, and make fair comparisons between, these two design

Table 1. Area comparison of NCL_D and NCL_X design flows.

	Area (transistor count)		Reduction
Circuit	NCL_D	NCL_X	(%)
AND4	96	30	69
AND16	480	186	61
BIT_CNT	2,779	2,432	12
CLIPPER	448	237	47
FA	176	118	33
FSM_DataPATH	10,260	9,290	9
HA	72	56	22
IF_THEN_ELSE	72	38	47
MUX_DECODER	456	352	23
NCL_ADDRCONV	1,356	1,045	23
NCL_X1	18,606	12,724	32
NCL_X2	15,338	13,852	10
SERIAL_CRC	2,010	1,936	4
SET_CNT	490	456	7
SYNC_STATE	2,402	1,934	19
USHIFT	1,264	762	40
VITERBI DECODER	45,198	38,130	16
Average			28

Table 2. Energy comparison of circuits produced by NCL_D and NCL_X design flows.

	Energy (%) for V_{DD} = 2.5 V		Energy (%) for V_{DD} = 1.8 V		Energy (%) for V_{DD} = 1.1 V	
Design	NCL_D	NCL_X	NCL_D	NCL_X	NCL_D	NCL_X
Address	100	109	46	52	14	19
Data	100	90	55	46	12	5
Wavelet	100	102	47	50	38	29

flows. Table 1 lists the designs chosen as the benchmark suite for this comparison. Designs with large combinational-circuit components, such as AND4 (a four-input AND function), achieved the highest area reduction. Register-dominant designs, such as SET_CNT (a three-stage ring register that counts up to a certain number), achieved the least reduction. Area measurements represent the transistor count of physical cells in the threshold library. Based on this benchmark suite, NCL_X achieved an average area reduction of 28%.

Power consumption

The wavelet design comprises two data path blocks (address and data) and one control block (a finite-state machine). Table 2 gives results from PowerMill simulations of the NCL_D and NCL_X versions. The nominal supply voltage for the design is 2.5 V. However, the design is also fully operational under reduced supply voltages 1.8 V and 1.1 V. This additional advantage comes from the asynchronous implementation: Reducing power degrades performance but does not cause a design malfunction. Energy values are relative to the nominal supply voltage and to the NCL_D implementation.

According to Table 2, NCL_D and NCL_X designs consume about the same amount of power. We also confirmed this observation by analyzing the switching activities for NCL_D and NCL_X design styles. The additional switching in the NCL_X circuit comes from the completion network, but you can offset this by eliminating switches in the functional part.

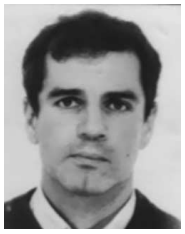
NCL_X IMPLEMENTATIONS have significantly lower area overhead, are faster, and consume approximately the same power as NCL_D implementations. However, compared with synchronous circuits, NCL_X implementations suffer from a 2- to 2.5-times area penalty and could consume more power. Power consumption might be less of an issue, because NCL_X implementations have natural support for idle mode and use a four-rail (instead of two-rail) communication scheme. But further reduction of the area penalty is unlikely, because it is close to a theoretical lower bound of 2 times the area penalty with respect to single-rail designs. Nevertheless, NCL_X circuits still have the advantages of extremely low noise and EMI, and higher levels of security and reliability during circuit operation. These advantages could make NCL_X circuits a good match for emerging mixed-signal and smart-card applications. ■

Acknowledgments

We thank Karl Fant, Michiel Ligthart, Ross Smith, and Alexander Taubin for their invaluable help in discussing and implementing the NCL_X design flow.

References

1. I. Sutherland and J. Lexau, "Designing Fast Asynchronous Circuits," *Proc. 7th Int'l Symp. Advanced Research in Asynchronous Circuits and Systems* (ASYNC 01), IEEE CS Press, Los Alamitos, Calif., 2001, pp. 184-193.
2. K. van Berkel et al., "A Fully-Asynchronous Low-Power Error Corrector for the DCC Player," *Proc. IEEE Int'l Solid-State Circuits Conf.* (ISSCC 94), IEEE Press, Piscataway, N.J., 1994, pp. 88-89.
3. S.B. Furber, D.A. Edwards, and J.D. Garside, "AMULET3: A 100 MIPS Asynchronous Embedded Processor," *Proc. Int'l Conf. Computer Design* (ICCD 00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 329-334.
4. J. McCardle and D. Chester, "Measuring an Asynchronous Processor's Power and Noise," *Proc. Synopsys Users Group Conf.* (SNUG 01), Synopsys, Mountain View, Calif., 2001, pp. 66-70.
5. D.E. Muller and W.S. Bartky, "A Theory of Asynchronous Circuits," *Proc. Int'l Symp. Theory of Switching*, Harvard Univ. Press, Cambridge, Mass., 1959, pp. 204-243.
6. A.J. Martin, "Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits," *Developments in Concurrency and Communication*, C.A.R. Hoare, ed., Addison-Wesley, Reading, Mass., 1990, pp. 1-64.
7. S.M. Nowick and D.L. Dill, "Automatic Synthesis of Locally Clocked Asynchronous State Machine," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 91), IEEE CS Press, Los Alamitos, Calif., 1991, pp. 318-321.
8. M. Ligthart et al., "Asynchronous Design Using Commercial HDL Synthesis Tools," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems* (ASYNC 00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 114-125.
9. K. Fant and S.A. Brandt, "Null Conventional Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis," *Proc. Int'l Conf. Application-Specific Systems, Architectures, and Processors* (ASAP 96), IEEE CS Press, Los Alamitos, Calif., 1996, pp. 261-273.
10. I.E. Sutherland, "Micropipelines," *Comm. ACM*, vol. 32, no. 6, June 1989, pp. 720-738.
11. I. Blunno and L. Lavagno, "Automated Synthesis of Micro-Pipelines from Behavioral Verilog HDL," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems* (ASYNC 00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 84-92.
12. J. Sparso and J. Staunstrup, "Delay-Insensitive Multi-Ring Structures," *Integration, the VLSI J.*, vol. 15, no. 3, Oct. 1993, pp. 313-340.



Alex Kondratyev is a research scientist at Cadence Berkeley Laboratories. His research interests include asynchronous design, theory of concurrency, and embedded systems. Kondratyev has an MS and PhD in computer science from the Electrotechnical University of St. Petersburg, Russia. He is a senior member of the IEEE and a member of the ACM.

Kelvin Lwin is an R&D engineer at Reshape Inc. His research interests include asynchronous design automation and system-on-a-chip design integration. Lwin has a BS in electrical engineering and computer science from the University of California, Berkeley.

Direct questions and comments about this article to Alex Kondratyev, Cadence Berkeley Laboratories, 2001 Addison Street, 3rd floor, Berkeley, CA 94704; kalex@cadence.com.