Design of asynchronous circuits by Synchronous CAD tools

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

Kelvin Lwin
Reshape Inc.
1255 Terra Bella Ave.
Mountain View, CA 94043, USA
klwin@reshape.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 features are far behind synchronous commercial tools. This paper considers a particular subclass of asynchronous circuits (Null Convention Logic or NCL) and suggests a design flow that is based entirely on commercial CAD tools. This new design flow shows a significant area improvement over known flows based on NCL.

1. INTRODUCTION
EDA flows, being industry-driven, use synchronous methodology as a de-facto standard. However, the implementation problems presented by imposing a synchronous model of operation in deep-submicron circuits motivates the investigation of other modes of operation, asynchronous in particular [1].

Asynchronous design has been proven capable of delivering:

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

However the success stories in high-speed and low power asynchronous designs are somewhat controversial. To deliver the promised advantages they often rely on non-trivial timing assumptions that make verification difficult. Moreover, a lack of commercial CAD support for asynchronous synthesis is a major roadblock to wider acceptance of the methodology.

Low EMI and noise coefficients are the only "free" advantages of asynchronous circuits. Getting rid of the clock results in a significantly flatter noise/EMI spectrum across the frequency domain (10DB drop according to [4]). Until recently, EMI and noise metrics were "second class citizens." The focus was on power and performance, but that is about to change. EMI and noise metrics are gaining significance because of two emerging applications: mixed signal and smart cards. For mixed signal designs, analog blocks are sensitive to clock correlated, digital switching noise. Reducing noise and EMI has an immediate impact, boosting both precision and performance significantly.

In a smart card domain, functionality is not sensitive to EMI itself, but the security is. Non-invasive security attacks are based on monitoring the power rail, or EMI signature, of a smartcard. Even distribution of circuit-switching activities vastly improves the security.

This paper suggests an automatic flow for the design of asynchronous circuits featuring: low EMI, high security and small flow turnover cost (HDL based methodology using commercial CAD tools) at a significantly reduced area penalty than the previous flow [5].

Section 2 introduces main theoretical notions. Section 3 describes a previously known HDL flow for NCL. Section 4 suggests a new way of NCL implementation. Section 5 presents experimental results for the suggested flow.

2. THEORTICAL BACKGROUND
2.1 Delay-Insensitive Combinational Circuits
The acknowledgement notion plays a key role in ensuring delay-insensitivity (DI). Informally, we say that the firing of gate g_i **acknowledges** the firing of gate g_j**,** if by looking at g_i switching, one can judge that g_j has already fired as well. In a delay-insensitive combinational circuit, any transition at the wire OR gate must be acknowledged by the primary outputs.

In practice, implementing the acknowledgment of every single wire in a circuit is costly. One can consider some wire forks in a circuit to be safe by making a timing assumption about their skew. These forks are called **isochronic** [6]. For an isochronic fork, it is sufficient to acknowledge a single wire from the fork while the acknowledgements for the rest of wires rely on the timing assumption. Circuits in which the only non-acknowledged wires come from isochronic forks are called quasi-delay-insensitive [6]. NCL circuits belong to the QDI class.

2.2 Null Convention Logic (NCL)
NCL is a specific way of implementing data communication based on **DI encoding.** Data changes from the spacer (**NULL**) to a proper codeword (**DATA**) in the set phase, and then back to NULL in the reset phase. NCL targets the simple DI encoding in which DATA codewords are **one-hot codes,** and the spacer

NULL is represented by a vector with all entries equal to "0". For example in **dual-rail encoding** each signal *a* is represented by two wires *a.0* and *a.1* (i.e. *a*=1 encoded as *a.0*=0, *a.1*=1, and *a=0* encoded as *a.0*=1, *a.1*=0).
At an architectural level, NCL systems show a clear separation of sequential and combinational parts, much *in the same way as with synchronous systems* (see Figure 1).

NCL systems borrow the idea of organizing register interaction and data communication in DI fashion from micropipeline architectures [7].

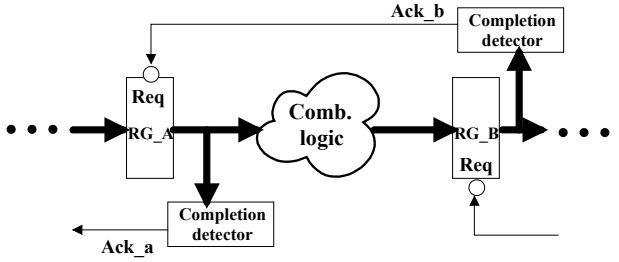

Figure 1. NCL system implementation

To explain how the NCL system functions, let us assume that all registers are initially in the NULL state and signals Ack are asserted to "0" (signals *Req* are asserted to "1"). When DATA arrives, the outputs of a register (e.g. RG_A) will change 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 (RG_B). Simultaneously, a completion detector checks for a DATA codeword at its inputs, and replies by rising 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 [7] ensures a **two-phase discipline** in NCL system functioning and prevents collisions between different DATA wavefronts.

This behavior scales down to the level of NCL gates. Every gate implements the so-called threshold function and is represented as $g(x_1,\ldots,x_n) = S + g(x_1+x_2+\ldots+x_n)$, where *S* is an unate set function. A gate *g* switches from NULL to DATA when its set function turns to 1, and it resets to NULL back when all inputs are reset to 0. 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.

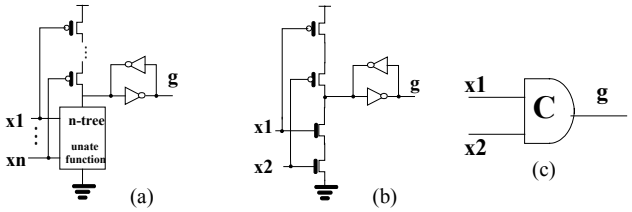

Figure 2. NCL gate implementations

3. Overview of HDL Design Flow

The NCL design flow uses off-the-shelf simulation and synthesis components (see Figure 3). 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 in GTECH library as if it is a conventional Boolean RTL. The second step expands the intermediate GTECH netlist into a dual-rail NCL by making dual-rail expansions and mapping into the threshold library. The particulars of implementation on step1 and step 2 impact the quality of final results.

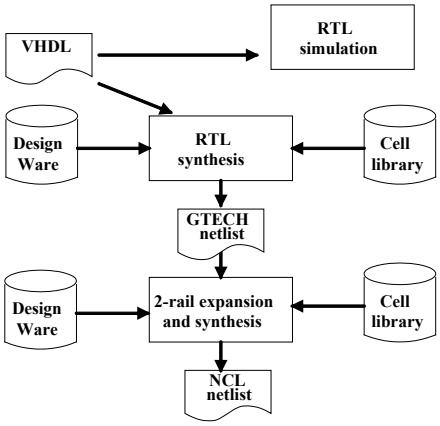

Figure 3 . RTL flow for NCL

In [5] a regular method for NCL implementation based on Delay Insensitive Minterm Synthesis (DIMS) [8] was suggested. In this method, steps 1 and 2 of the design flow are implemented as follows:

Step 1 performs a mapping of the optimized network into two-input NAND, NOR and XOR gates.

Step 2 first represents each wire *a* as a dual-rail pair *a.0* and *a.1* and then makes a direct translation of two-input Boolean gates into pairs of threshold gates with limited optimization of a threshold network.

Unfortunately DIMS-based implementations have significant overhead that comes from two main sources:

1. Overdesigning due to locality of ensuring DI (no sharing in the acknowledgement is allowed)
2. Little room for optimisation (optimisation can easily destroy DI properties)

4. NCL Flow with Explicit Completeness

The newly proposed flow exploits the idea of separate implementation of functionality and delay-insensitivity. A NCL circuit is partitioned on functional and completion parts with the possibility to optimize them independently from each other. This is achieved by a following modification of flow steps 1 and 2.
Step 1 performs a conventional logic synthesis (with optimization) from RTL specification of NCL. It maps an obtained network into GTECH library that consists of gates implementing set functions of threshold gates.

Step 2 consists of the following substeps:

2.1. Reduction of 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.

2.2. Dual-rail expansion of the combinational logic by creating for each gates in the *rail.1* network its corresponding dual gate in *rail.0* network.

2.3. Ensuring delay-insensitivity by providing local completion detectors (OR gates) for each pair of dual gates and connecting them into a completion network (multi-input C-element) with a single output *done.*

Implementation of the flow with explicit completion requires a minor modification of interfacing conventions within the NCL system. From now on we will assume that for each 2-rail primary input *a.0, a.1* there exists an explicit signal *a.go* such that $a.0 \neq a.1 \Rightarrow a.go=1$ (set phase), while $a.0=a.1=0 \Rightarrow a.go=0$ (reset phase). The modified organization of NCL system is shown in Figure 4. It is easy to see that it differs from the one in Figure 1 in having separate completion detectors for combinational logic and registers.

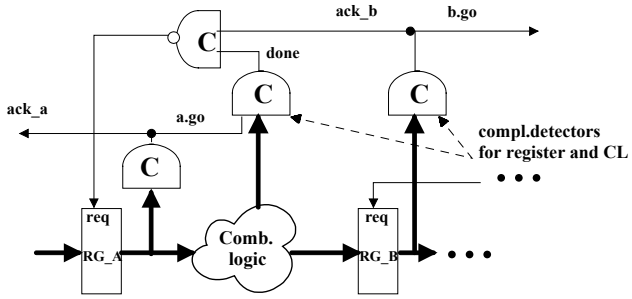

Figure 4. NCL system with explicit completion detection

Example. **Encoder 4 to 2.** The encoder is described by the following RTL specification:

```
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;
```

A mapping of encoder in GTECH library (step 1) is shown in Figure 5(a). After the reduction to unate gates (Figure 5(b)) the network is expanded into dual-rail implementation (Figure 5(c)). This is done locally by providing a dual function for each of the gates in unate representation from Figure 5(b). Each dual-gates pair contributes an output by a local completion detector (OR gate) that goes to multi-input C-element, which is providing signal *done* (Figure 5(d)). Information about the validity of input codewords is provided by the C-element, *I.go*. It combines all completion signals for primary inputs. The output *I.go* is connected to the C-element *done*.

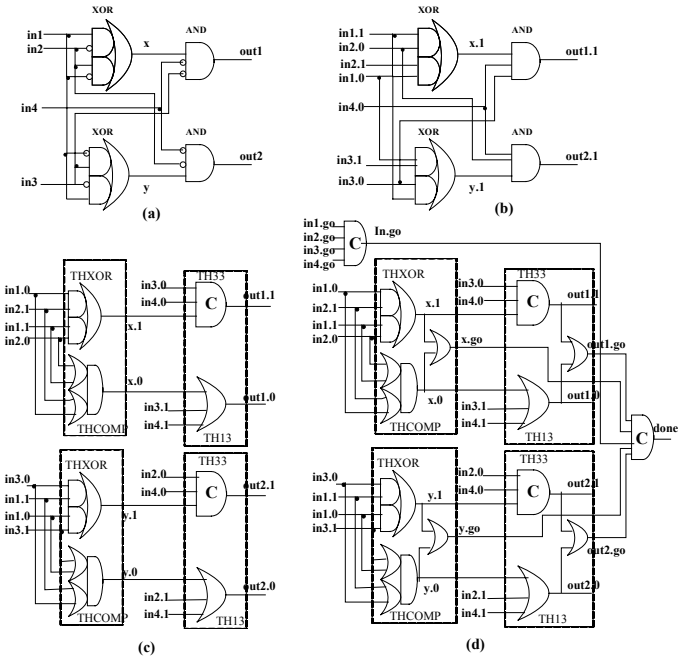

Figure 5. 4 to 2 encoder with explicit completion

Proposition. A NCL circuit implemented with explicit completion belongs to QDI class.

Proof for the proposition at hand is rather straightforward. In each phase (set and reset) exactly one gate out of the dual pair must switch. These transitions are acknowledged by local completion detectors. In their own turn all local detectors are acknowledged by the primary output *done.* Hence, at least one wire in every fork is acknowledged and a circuit is from QDI class.

5. EXPERIMENTAL RESULTS

This Section provides experimental data for comparison of the suggested flow with the previously known approach [5] according to area and power consumption. Speed estimation was less of an issue because the flow does not target high-speed applications. However preliminary data show that NCL_X circuits do not show degradation of performance with respect to [5].

Area. Implementation of NCL flow with explicit completion (NCL_X flow) kept the same front end RTL coding style so it is compatible with the known NCL flow based on DIMS [8] (NCL_D flow). This enables the same designs accomplished in NCL_D to be synthesized using NCL_X. Between these two, fair comparisons can be conducted. Table 1 illustrates the designs chosen as benchmark suite to compare the relative performance of NCL_X versus NCL_D. It shows the designs with large combinational clouds to have achieved the highest reduction in area (e.g. AND4) and registration elements dominant designs showing very little reduction (e.g. SET_CNT). The AND4 design is merely a 4-input AND function implementation. The circuit SET_CNT is a 3-stage ring register that counts up to a certain number. The area number used for the figures is based on the transistor count of physical cells in the threshold library. According to the design suite, the new flow, NCL_X, can achieve an average area reduction of 28%.

Table 1. Area Comparison

Circuit	NCL_D	NCL_X	%
AND16	480	186	-61%
AND4	96	30	-69%
IF_THEN_ELSE	72	38	-47%
HA	72	56	-22%
FA	176	118	-33%
CLIPPER	448	237	-47%
MUX_DECODER	456	352	-23%
SET_CNT	490	456	-7%
USHIFT	1264	762	-40%
NCL_ADDRCONV	1356	1045	-23%
SERIAL_CRC	2010	1936	-4%
SYNC_STATE	2402	1934	-19%
BIT_CNT	2779	2432	-12%
FSM_DATAPATH	10260	9290	-9%
NCL_X2	15338	13852	-10%
NCL_X1	18606	12724	-32%
VITERBI DECODER	45198	38130	-16%
Average	**-**	**-**	**-28%**

Power consumption. The wavelet design consists of two datapath blocks address and data) and one control block (FSM). Results from PowerMill simulations of NCL_D and NCL_X versions are shown in Table 2. The nominal supply voltage for the design is 2.5V. However the design is fully operational under the reduced supply voltages 1.8V and 1.1V. This is an additional advantage that comes from the asynchronous nature of the implementation in which reducing power results in performance degradation but not in design malfunction. Energy numbers are presented relative to the nominal supply voltage and NCL_D style of implementation. From Table 2 follows that NCL_D and NCL_X designs show approximately the same power numbers. This conclusion was also confirmed by analysis of switching activities for NCL_D and NCL_X design styles. For NCL_X circuit, the additional switching comes from completion network but in general offset by having smaller number of switching in the functional part.

Table 2. Energy Comparison

Design	Energy (V_{dd}=2.5V)		Energy (V_{dd}=1.8V)		Energy (V_{dd}=1.1V)	
	NCL-D	NCL-X	NCL-D	NCL-X	NCL-D	NCL-X
addr	100%	109%	46%	52%	14%	19%
data	100%	90%	55%	46%	12%	5%
wavelet	100%	102%	47%	50%	38%	29%

6. CONCLUSION

The paper has presented an efficient design flow for asynchronous circuits that is fully supported by commercial CAD tools. Implementations obtained through the flow are compared favorably to the ones reported in [5]: they show significantly lower area overhead, are faster and consume approximately the same power. To be fair we should admit that in comparison to synchronous circuits NCL_X implementations suffers from 2-2.5 times the area penalty and could consume more power. Power consumption might be less of the issue because it could be reduced due to natural support of idle mode and through the use of four-rail communication scheme instead of two-rail. Area penalty is unlikely to be further reduced because it is close to a theoretical lower bound of 2X coming from dual-rail nature of designs. However the advantages of NCL_X circuits with respect to synchronous are extremely low noise and EMI [4] and higher level of security and reliability during the circuit operation. Due to these advantages NCL_X circuits might be a good for emerging mixed signal and smart card applications.

7. ACKNOWLEDGMENTS

We wish to thank Alexander Taubin, Ross Smith and Michiel Lighthart for their invaluable help with the NCL_X flow implementation.

8. REFERENCES

1. International Technology Roadmap for Semiconductors, http://www.itrs.net/

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. 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.

4. John McCardle, David Chester. Measuring an Asynchronous Processor's Power and Noise, SNUG Conference, April 2001.

5. 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.

6. A.J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, ed., Developments in Concurrency and Communication, UT Year of Programming Series, pages 1-64. Addison-Wesley, 1990.

7. Ivan E. Sutherland. Micropipelines. Communications of the ACM, 32(6):720-738, June 1989.

8. Jens Sparso and Jorgen Staunstrup. Delay-insensitive multi-ring structures. Integration, the VLSI journal, 15(3):313-340, October 1993.