High-Level Design for Asynchronous Logic

Ross Smith, Michiel Ligthart
Theseus Logic
{ross.smith, michiel.ligthart}@theseus.com

Abstract

Asynchronous, self-timed, logic is often eschewed in digital design because of its ad-hoc methodologies and lack of available design tools. This paper describes a complete High Level Design flow for asynchronous circuits based on Register Transfer Level (RTL) VHDL using commercial simulation and synthesis tools. Contrary to previous asynchronous approaches, the proposed RTL methodology closely resembles familiar synchronous design styles.

1.0 Introduction

In 1997, the Semiconductor Industry Association (SIA) identified a major design crisis approaching in its "National Technology Road Map for Semiconductors" [Sia97]. Specifically, they mentioned the drastic increase of clock frequency and the exponential increase of circuit complexity, and concluded that a novel asynchronous approach could be required to solve some of the industry's future problems. Additionally, the trend towards Systems-On-Chip (SOC) technology may very well encourage the use of asynchronous design in the electronics industry. The prime goal of SOC technology is to create truly reusable Intellectual Property (IP) blocks that can be built quickly and are guaranteed to work the first time. Together with high quality IP pieces, an overall design methodology should provide a simple means for IP assembly, based on plug and play principles. Clock free, asynchronous circuits constitute an attractive SOC approach. From the system's architecture point of view, it is much easier to build SOCs using asynchronous rather than synchronous blocks where multiple clock domains need to be interfaced.

As it turns out, several promising asynchronous design methodologies have been proposed over the past ten years. Excellent overviews of the state-of-the-art can be found in [Berkel99], [Hauck95] and [Nanya93]. However, most of these asynchronous design methodologies lack easily accessible, standard High Level Design tools based on conventional Hardware Description Languages.

This paper introduces a complete High Level Design methodology for asynchronous, self-timed circuits based on a Register Transfer Level (RTL) description in VHDL[1] as illustrated in figure 1. The paper is organized as follows. We start with a description of an asynchronous design methodology called NULL Convention Logic (NCL) and its relation to other delay-insensitive techniques. Next, we describe an RTL simulation environment implemented with off-the-shelf IEEE 1076/1164 compliant VHDL simulators. We continue with an explanation of how this HDL description is used with a commercial Logic Synthesis tool such as Design Compiler to synthesize an optimized, asynchronous netlist ready for Place and Route. The result section compares several manual designs with synthesized results, as well as synthesized asynchronous designs with synthesized synchronous designs.

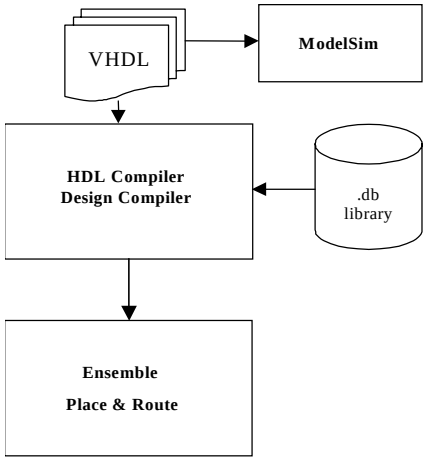

Figure 1: HLD flow for asynchronous design in NCL.

2.0 NULL Convention Logic

NULL Convention Logic (NCL) [Fant96] is a logic that yields inherently asynchronous, delay insensitive circuits and systems. In an NCL design each gate is a synchronization node, managing the interaction of alternating DATA and NULL wavefronts. DATA wavefronts contain the computational information

[1] The approach is also applicable to Verilog HDL.

and are processed by the gates to perform the desired data transformation. NULL wavefronts do not contain any computational information and are used to separate successive DATA wavefronts. The wavefront flow and interaction is managed through an asynchronous acknowledge / request signaling protocol. This request-acknowledgement protocol is illustrated in figure 2. When the outputs of the circuit are all DATA, the completion detection circuit signals a request for NULL to the inputs. When all outputs of the circuit are at NULL, a request for new DATA is issued by the same completion detection circuitry.

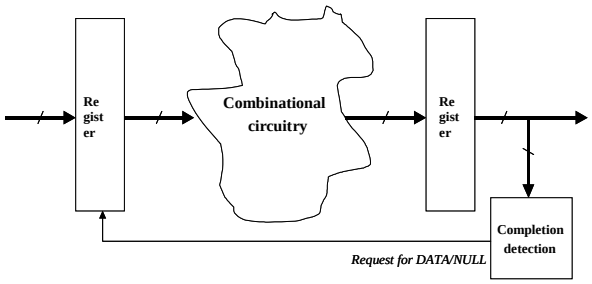

Figure 2. Basic NCL circuit with REQ/ACK.

In NCL, this behavior is pushed down to the level of each particular gate of a circuit. If the current ouput of a gate is NULL then the gate keeps its output at NULL as long as NULL is present at any one of its inputs. When all gate inputs receive DATA, the output of the gate changes to DATA. The output then maintains DATA until all inputs receive NULL before changing to NULL.

This behavior is naturally expressed in a multi-valued logic with '1' and '0' as DATA values and 'N' for NULL. The behavior of a 2-input gate in NCL is shown in figure 3. Notice how the NCL gate switches differently depending on the current value of the output. This sequential behavior of the gates is referred to as *hysteresis*.

current value of output Z is NULL		
a	**b**	**z**
D	D	D
D	N	N
N	D	N
N	N	N

current value of output Z is DATA		
a	**b**	**z**
D	D	D
D	N	D
N	D	D
N	N	N

Figure 3. Symbolic tables for 2-input NCL gates.

The representation of NCL gates in a three-level logic is called 3NCL. Although 3NCL is a convenient mathematical abstraction, it has no efficient physical implementation due to the binary nature of signals used in design practice. For physical implementation each signal *a* in 3NCL is represented by two wires *a.rail1* and *a.rail0* in a circuit under the following encoding of 3NCL symbolic values:

```
a='1' ⇔ a.rail1='1', a.rail0='0';
a='0' ⇔ a.rail1='0', a.rail0='1';
a='N' ⇔ a.rail1='0', a.rail0='0'.
```

The combination of values `a.rail1 = '1', a.rail0 = '1'` is not used. This encoding is known as a dual-rail encoding [Sims58].

Implementation of 3NCL logic through a dual-rail encoding, called 2NCL, gives a physical representation of NCL. Sequential behavior of a gate in 2NCL is ensured through a feedback from the gate's output to its inputs, which allows representation of the gate's behavior by the logic equation

$$g = S + g\overline{R}$$

where *S* and *R* are the set and reset functions of the gate. A general view on semi-static CMOS implementations of gates in 2NCL is shown in figure 4a.

A refined picture of a gate's structure is obtained through consideration of specific properties of dual-rail circuits under two-phase (set and reset) operation. These properties are:

1.0 In a dual-rail circuit a transition from NULL to DATA is monotonic; and
2.0 The transition of primary inputs of a combinational circuit from DATA to NULL results in the setting of all gates in a circuit into the NULL state

From (1) it follows that a set function S of a gate must be positive unate [Brayton90]. Set conditions for NCL gates are conveniently specified with threshold functions, a particular subclass of unate functions. A threshold function S is one that can be defined by a system of inequalities: $S(x_1,\ldots,x_n) = 1$ *iff* $w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq m$, where w_i are the weights, m is the threshold value and "+" is an arithmetic sum. In the case where all weights are equal to '1' a threshold function can be characterized by two numbers: i) n – number of inputs, and ii) m –

the threshold value. This simplified representation is called an *m*-of-*n* threshold function.

The reset function for an NCL gate follows from point (2). An NCL gate changes its output to NULL when all its inputs are NULL. Bearing in mind that DATA values at inputs of a gate are encoded by "01" or "10" we arrive at

$$R(x_1,\ldots,x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$$

A refined view on the implementation of a 2NCL gate is shown in figure 4(b). In this paper, we refer to this implementation as a threshold gate with hysteresis. Actual CMOS implementations of these kinds of gates are described in [Sobelman98].

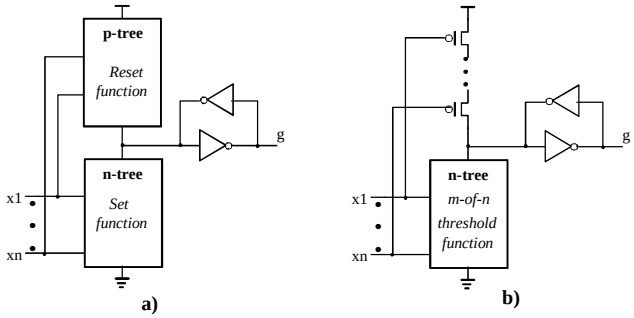

Figure 4. Implementation of an NCL gate in CMOS.

3.0 RTL design for NCL

RTL descriptions of asynchronous designs should closely match common synchronous description styles, with minor adjustments to account for the asynchronous behavior. For NCL this means the RTL description style has to account for:

- NULL/DATA behavior
- Hysteresis
- Asynchronous registers with REQ/ACK signals

In addition, to stay at an acceptable level of abstraction, designs should be described in 3NCL (multi-value abstraction) rather than 2NCL (dual-rail implementation).

These requirements are met with the following HDL coding rules:

- Introduce a 3-valued logic type with values {'N', '1', '0'}.
- Use a simulation-only assignment inside processes to describe the NULL/DATA and hysteresis behavior.
- Rely on logic synthesis to expand signals into their dual-rail equivalents.
- Use a pre-defined and pre-mapped function call for the req/ack circuit generation used with asynchronous registers.

These rules allow the designer to use a familiar style of VHDL (if-then-else, case, arithmetic and relational operators, process as well as dataflow assignments, register inferencing, etc.). An example of a 3NCL circuit described in VHDL is shown in figure 5.

```
library ncl;
use ncl.ncl_logic.all;
entity example is
   port (a,b,s : in  ncl_logic;
         z     : out ncl_logic );
end example;
architecture ncl of example is begin
   process (a,b,s) begin
      if s = '1' then
         z <= a;
      else
         z <= b;
      end if;
      hysteresis(a,b,s,z);
   end process;
end ncl;
```

Figure 5. Example of a 3NCL description in VHDL.

The following three sub-sections discuss the simulation and synthesis implications of this approach.

3.1 Simulation

During simulation, NULL/DATA behavior and hysteresis are modeled with an NCL-specific simulation package 'ncl_logic'.

The package defines a type NCL_LOGIC, completely analogous to the IEEE 1076/1164 STD_LOGIC type, with the addition of a new value 'N' for NULL:

```
type ncl_logic IS (
     'U','X','0','1','N','Z','-');
type ncl_logic_vector is array (
   natural range <>) OF ncl_logic;
```

Based on this type, all basic Boolean functions (and, nand, or, nor, etc.) as well as conversion functions defined in the IEEE 1076/1164 'std_logic' package are overloaded for NCL_LOGIC and NCL_LOGIC_VECTOR.
Hysteresis functionality is provided through a pre-defined procedure, which is included in the 'ncl_logic' package. This procedure is called at the end of every process or dataflow statement with all input and output process signals as parameters. Because hysteresis is an inherent characteristic of NCL's threshold gates, it should not be synthesized. Therefore, the hysteresis procedure is surrounded by 'synthesis on/off' pragmas. In pseudo-VHDL code, this looks as follows:

```
hysteresis (input_signals, output_signal)
-- synthesis_off
   if (all input_signals = 'N') then
      output_signal <= 'N';
   elsif (any input_signals = 'N') then
      output_signal <= output_signal;
   end if;
-- synthesis_on
```

The function causes the output signal to be modified only when the inputs are either all NULL or all DATA. The hysteresis function is invoked as the last statement in a process as shown in figure 5, or on concurrent signal assignments as follows :

```
z <= a and b when hysteresis(a&b)
                    else unaffected;
```

Analogous to STD_LOGIC, NCL_LOGIC also has support for arithmetic packages

- NCL_ARITH
- NCL_SIGNED
- NCL_UNSIGNED

with full support for all arithmetic and relational operators.

With this functionality, we can now describe 3NCL circuits in VHDL and simulate them with any IEEE 1076/1164 compliant simulator. For improved performance, NCL packages can be pre-compiled and linked with the simulator just like STD packages.

3.2 Synthesis

In order to use conventional logic synthesis tools for NCL, the flow has to handle:

- The 'N' value in NCL_LOGIC
- Hysteresis
- Asynchronous registers with ACK/REQ signals

The first issue can be handled by treating the 'N' value in NCL_LOGIC as a '-' (don't care) value. This enables tools such as Design Compiler to treat a 3NCL variable as a single wire. Hence, the same 3NCL description used for simulation purposes can now be used as input for logic synthesis.

Hysteresis is completely ignored during logic optimization and technology mapping. Because every node in the network has hysteresis, the combination of two nodes with hysteresis results in a new node with hysteresis. Likewise, if a node with hysteresis is split into two nodes, the two nodes will both have hysteresis. Hysteresis is also ignored during technology mapping, because every library cell has hysteresis. If a node in a network is mapped onto one or more cells from a library, each mapped cell will have hysteresis. Therefore, the functionality of the cells in the library is described without hysteresis as well.

Asynchronous registers can be inferenced with an incompletely specified `'if (condition) then (assignment)'` statement, which synchronous synthesis tools map to a D-LATCH primitive. For NCL purposes, the primitive is overloaded with an asynchronous register with pins IN, OUT, and REQ (RST is optional). Completion detection, which produces Request and Acknowledge signals, is provided through a pre-defined function call CMPD() as illustrated in figure 6.

```
process (req, a, b, z) begin
   if (req = '1') then
      z <= a + b;
   end if;
   zack <= cmpd(z);
   hysteresis (req, a, b, z);
end process;
```

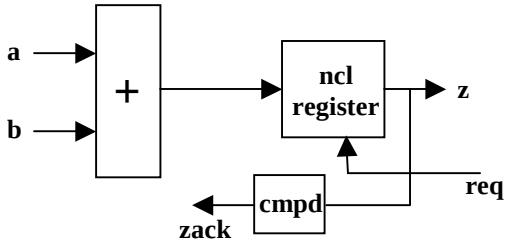

Figure 6. HDL code and synthesized circuit of inferred asynchronous register.

By ignoring the 'N' value and hysteresis, logic synthesis for NCL is reduced to a standard synthesis

problem that can use commercial synthesis tools. The logic synthesis for NCL is implemented in two steps (see also figure 7):

1) Translate 3NCL RTL into a 3NCL netlist

The circuit description in 3NCL, using the multi-valued NCL_LOGIC type, is input to a commercial synthesis tool, e.g., Design Compiler. The synthesis tool optimizes the HDL and maps it to a generic Boolean library. Dataflow components, such as adders, incrementors, and comparators are mapped on built-in DesignWare components. The resulting netlist is referred to as a 3NCL netlist. Note that the signals in this 3NCL netlist are still of the type NCL_LOGIC, i.e., can carry values '0', '1', and 'N'.

2) Translate 3NCL into a 2NCL netlist

In the second step, the intermediate 3NCL netlist is expanded into a fully dual-rail 2NCL netlist by redefining all signals as dual-rail signals, and overloading all generic Boolean components (AND, OR, XOR, INV, MUX etc.) with dual-rail equivalents. The actual expansion is described in a VHDL package that is read before the 3NCL code from step 1 is read. Like step 1, the 2NCL transformation in step 2 is done with a commercial synthesis tool. At this point, the original 3NCL behavioral description has been transformed into a fully dual-rail gate level representation, with preservation of arithmetic functions.

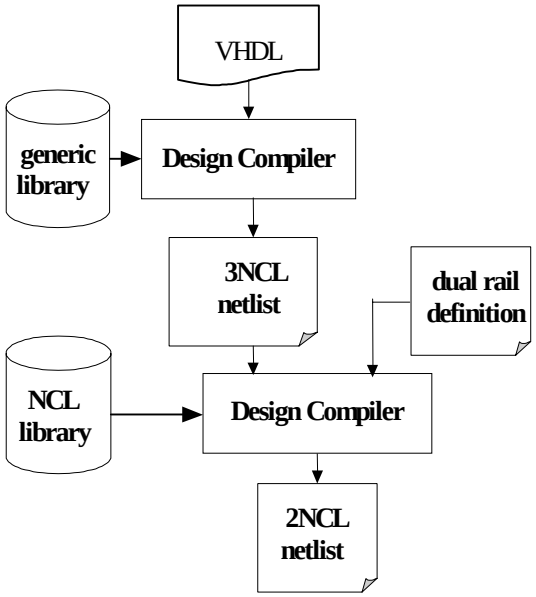

Figure 7. Two-step implementation of logic synthesis for NCL.

Step 2 subsequently reads in a standard synthesis library with threshold gates and performs ASIC-type optimization and technology mapping.

The dual rail definition is provided in a package which is read in together with the intermediate 3NCL netlist. This package describes a dual rail signal as a record and overloads the basic components for dual rail implementations:

```
type dual_rail_logic is record
       rail1 : std_logic ;
       rail0 : std_logic ;
end record;

function "and" (a,b: dual_rail_logic)
              return r: dual_rail_logic;
   r.rail1 <= a.rail1 and b.rail1;
   r.rail0 <= (a.rail0 and b.rail0) or
              (a.rail0 and b.rail1) or
              (a.rail1 and b.rail0);
end and;
```

3.3 Design Compiler scripts and libraries

The flow described above has been implemented in Design Compiler (DC) scripts and libraries that are transparent to the designer. An NCL library is identical to any other .db library compiled for DC and contains gate functionality, timing parameters, and wire load models.

Setting up DC to run with NCL and perform the actual dual rail expansion is implemented in two scripts 'ncl_init' and 'ncl_compile'. Figure 8 shows how a VHDL description is mapped and optimized for NCL in DC. NCL commands are in italic typeface.

```
> ncl_init
> read –f vhdl ifthen.vhd
> ncl_compile
> read –f db ncl_25.db
> target_library = ncl_25.db
> link_library = ncl_25.db
> symbol_library = generic.sbd
> compile
```

Figure 8. Design Compiler script for NCL circuit synthesis.

4.0 Results

Table 1 compares manual NCL design using schematic capture with synthesized NCL designs. The synthesized designs were written in VHDL and optimized with DC. As expected, synthesis generates

excellent results compared with manual design. Area results are in number of transistors.

design	manual	synthesis	ratio
if-then-else	40	40	100 %
and4	66	68	103 %
test7	140	122	87 %
clipper	339	208	61 %
set_cnt	238	202	85 %
and16	352	336	95 %
shift	506	284	56 %
case	594	482	81 %
sync_state	1008	814	81 %
bit_cnt	1059	1072	101 %

Table 1: Comparison of manual and synthesized asynchronous designs (area is in transistors).

Table 2 compares synthesized NCL circuits with synthesized clocked circuits, mapped on an equivalent LCB500k library [Lsi95]. 'X2vhd' is a 16-bit arithmetic limit/round operation that is purely combinational and hence the synchronous and asynchronous VHDL descriptions are identical. The 'decoder' example is a Viterbi decoder where the asynchronous VHDL description is a straight translation from the synchronous VHDL. The, 'hostfird' example is a thirty-two by 16-bit unidirectional FIFO buffer which was redesigned for NCL. The gate count is the number of actual library gates. Notice that the gate count of 'hostfird', which was redesigned for NCL, is comparable to the gate count of the synchronous design, whereas straight re-implementations done for 'X2vhd' and 'decoder' generate much larger designs.

	number of gates		
module	synchr	NCL	ratio
addrconv	41	62	1.5
X2vhd	395	826	2.1
decoder	1010	1804	1.8
hostfird	1691	1123	0.7

Table 2. Comparison of synthesized synchronous and NCL designs.

5.0 Summary

In this paper, we have introduced a flow to simulate and synthesize asynchronous, delay-insensitive circuits with commercial EDA tools identical to the flow used for synchronous circuits. An RTL description of an asynchronous circuit in VHDL can be simulated with any IEEE 1076/1164 compliant simulator and synthesized with a commercial synthesis tool such as Design Compiler. The resulting netlist, mapped on a standard cell library of threshold gates, is ready for automatic place and route. The availability of this flow will make asynchronous circuit design easier to adopt.

Acknowledgements

Many people provided us with invaluable feedback during the development of our NCL simulation and synthesis flow. Special thanks go to Kelvin Lwin and Tam Nguyen for implementing the packages and running the experiments. Alexander Taubin and Alex Kondratyev contributed the section on NCL.

References

[Berkel99] 'Special issue on asynchronous circuits and systems' C. H. (Kees) van Berkel, Mark B. Josephs, and Steven M. Nowick, eds., *Proceedings of the IEEE*, 87(2), February 1999.

[Brayton90] R. K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, Vol.78, No.2, February 1990, pp. 264-300

[Fant96] Karl M. Fant and Scott A. Brandt. NULL conventional logic: A complete and consistent logic for asynchronous digital circuit synthesis. *International Conference on Application-specific Systems, Architectures, and Processors*, pages 261-273, 1996.

[Hauck95] Scott Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1): 69-93, January 1995.

[Lsi95] LCB500K Preliminary Design Manual, LSI Logic Corporation, Milpitas, CA, 1995.

[Nanya93] Takashi Nanya, 'Challenges to dependable asynchronous processor design', in *Logic Synthesis and Optimization*, Tsutomu Sasao, editor, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[Sia97] Semiconductor Industry Association, *"The National Technology Road Map for Semiconductors'*, 1997 Edition

[Sims58] C. Sims and H. J. Gray. Design criteria for autosynchronous circuits. *Proc. Eastern Joint Computer Conf. (AFIPS)*, volume 14, pages 94-99, December 1958.

[Sobelman98] Gerald E. Sobelman and Karl Fant. CMOS circuit design of threshold gates with hysteresis. *Proc. International Symposium on Circuits and Systems*, pages 61-64, June 1998.