Asynchronous Quasi Delay Insensitive (QDI) Circuits:
Infrastructure, Templates, Design and Analysis

Ney L. V. Calazans, Matheus T. Moreira, Marcos L. L. Sartori, Alexandre M. Amory, Felipe Kuentzer
What is a Synchronous Circuit?

General Structure of a Synchronous Circuit

Storage

Inputs

Clock

Combinational Circuits

Outputs
What is a Synchronous Circuit?

Synchronous Operational Assumptions

- **Setup Time**: minimum time inputs stabilize before the clock edge
- **Hold Time**: minimum time inputs must remain stable after the clock edge

Setup Time – minimum time inputs stabilize before the clock edge

Hold Time – minimum time inputs must remain stable after the clock edge
What is a Synchronous Circuit?

The Clock Skew Concept

Clock waveform at point A

Clock waveform at point B

Clock signal skew
Wires are Changing...

In modern technologies
(Ho et al., The Future of Wires, Proc. IEEE, 2001)

State of the Art today: 7nm

- I.e. crossing a chip can take 5 clock cycles or more!
- Thus how can we employ na overall synchronous design?
- Avoiding this problem is increasingly expensive!!
- Besides, the clock distribution may take 60-70% of the overall chip power!!
- Consequence – large chips today are indeed GALS designs
Synchronous X Asynchronous

Advantages of Synchronous Design

1. Initial Design $\rightarrow$ Can abstract delays entirely
2. Design $\rightarrow$ Easy to decompose
3. Hazards (glitches) $\rightarrow$ Easy to deal with
4. Race problems $\rightarrow$ Easily solvable
5. Direct Boolean and algebraic manipulations
1. Clock skew in large chips → design nightmare!
2. Potential → Waste of energy
3. Worst case performance (Exceptions exist!)
4. Technology migration → Another nightmare!
5. Design → Can be inadaptable to physical properties variations → P, T, V, IR drop, etc.
6. Metastability Treatment → 3rd nightmare!
7. Digital ↔ Analog Integration → Hard
Synchronous X Asynchronous

1. Synchronous Design
   (i) A single method (a template, we’ll see soon)
   (ii) Well-defined
   (iii) Ample support by EDA tools

2. Asynchronous Design
   (i) Not one, dozens of design methods, or templates, exist
   (ii) Effective designs depend upon model choices
   (iii) Modeling choices depend on designer experience, which is rare...

In Short:
I. Synchronous design is powerful, but reaching its limits
II. Support to asynchronous design is missing, a lot!!
III. Research community/Industry are trying to fill the gap…
Currently in Async (QDI/BD)

• **Enterprises**
  - Tiempo Secure – Grenoble, France → QDI/BD Smartcards
  - Chronos Tech – San Diego, CA, USA → QDI communication
  - Galois – Portland, OR, USA → (REM) BD, Cryptography

• **Research Labs**
  - LETI Lab – Grenoble, France → QDI NoCs
  - IHP – Frankfurt (Oder), Germany → Resilient BD (USC)
  - PUCRS, Porto Alegre, RS, Brazil
  - Async design with Sync tools (Cadence/Synopsys)
  - **Wish list** – IoT ULP end nodes (subthreshold)
  - Several other places (Japan, Spain, UK, USA, Canada, etc.)
  - Async controllers, synchronizers for GALS, memories,…
Motivation

1. Introduction to Asynchronous Circuit (Design)
2. Infrastructure to Design Asynchronous Circuits
3. Models and Design Templates
4. QDI Design – Some advances
5. Async Analysis – Test of asynchronous circuits
1. Introduction to Asynchronous Circuit (Design)
Ok, we all know there are differences between sync and async designs, but what is the fundamental one?
The Synchronization Scheme!!

→ Implicit (global) synchronization between blocks

Clock Period  > Max Delay (CL) + R Delays (S/H)
The Synchronization Scheme!!

Asynchronous Circuits

→ Explicit synchronization: Req/Ack handshake, local to each R-CL pair (a stage)
→ No clock – performance depends on average timing
Before discussing async design methods, a need → a model for models (Metamodel)!

(Moreira, PhD, PUCRS, 2016) proposed a model for any design template → Not only for asynchronous!!
Applying the Metamodel to Synchronous

- Synchronous Circuits Design
  - The **Design Style** \(\rightarrow\) RTL
  - The **Set of Components** \(\rightarrow\)**
    Boolean Gates (e.g. standard cell or FPGA gate libraries)
  - **Architecture** \(\rightarrow\) (i) Interconnect gates, forming functional (CC) modules (data transformation); (ii) interconnect functional modules inputs and outputs to registers; (iii) connect register to inputs/outputs \(\rightarrow\) \(\rightarrow\)**
    Essence of register-transfer level (RTL)
  - The **Channel**
    - Communication link \(\rightarrow\)**
      wires encode **information** as **Boolean numbers**, clock controls everything, everywhere
    - **Protocol** \(\rightarrow\)**
      the **Synchronous** one, when the clock ticks, every register (or almost) gets new data, otherwise nothing changes
What can Change for Async?

• Almost nothing, or everything!
• The closer to synchronous, the easier to
  ‒ Understand
  ‒ Use sync tools to do async design
• The farther from synchronous, the better
  ‒ Power efficiency (… or not!)
  ‒ Robustness to variations, Single Event Effects (SEEs), technology migration
  ‒ Potential to age gracefully, produce less EMI!
Some Example Efforts

1. Desynchronization  (Cortadella et al., ITCAD, 2006)
   • Capture design synchronously
   • Do logic synthesis as usual
   • Do physical synthesis, changing clock tree step
   • Substitute clock by a set of local handshake controllers and delay lines (EDA)

2. Click elements  (Peeters et al., ASYNC, 2010)
   • Data-driven async circuits using power-saving, edge-based, local handshake comm. protocols
   • Only standard gates and FFs used → no special gates/libs needed
To Conclude this Part

• Moreira’s metamodel → Approach async design models and templates

• Several async templates → Set of Components distinct from Boolean gates (C-elements, NCL gates, multi-rail pseudo-dynamic gates, etc.)

• Basically, all async templates → Architecture different from sync

• Basically, all async templates → Channels different from sync

• Differences on Communication Link → Divide async templates on two big template families: BD and DI/QDI (more on this later on)
2. Infrastructure to Design Asynchronous Circuits
Why a New Flow for Libraries?

1. Asynchronous components are (mostly) sequential

2. Sequential cell characterization in commercial flows → tweak-based, not generic method
   Do not work (easily) for sequential cells distinct from latches and flip-flops!

3. Commercial tools for dimensioning transistors → not often allow async criteria to be considered

4. Asynchronous Standard Cells for ‘n’ Designs → ASCEnD
The ASCEnD-A Design Flow

(Karmazin et al., ASYNC, 2013)

(Prakash, MSc, USC, 2007)
ASCEnD-A: Templates

(Moreira, EofT, PUCRS, ASCEnD, 2010)
ASCEnD-A: Cell Sizing

(Moreira et al., ASCEnD-A, LASCAS 2012)
ASCEnD-A: Cell Layout

(Moreira et al., ASCEnD-A, SBCCI 2014)
(Ziesemer et al., ASTRAN/ASCEnD, GLSVLSI 2014)
ASCEnD-A: Cell Characterization

(Moreira & Calazans, LiChEn, ICECS 2012)
(Moreira et al., LiChEn, DSD 2013)
What Can be Done with ASCEnD?

1. Develop ordinary cells – Inverters, NANDs, etc.
2. Develop sequential ordinary cells like flip-flops
3. Create sequential, non-conventional cells
   - RoGen/CeS ➔ care for dimensioning C-elements, NCL gates
   - LiChEn ➔ enable characterizing any and all above (deals with non-conventional sequential, which commercial tools – Cadence/Synopsys - do not, at the time…)
4. Create cell libraries compatible with existing ones
   - Cell Library Templates Module exists in the flow
5. We created several libraries, designed and built circuits with them
The First ASCEnD Library (2011)

- ASCEnD – The first version
  - STM 65nm bulk CMOS
  - LIB & LEF
  - GP, Std Vth devices

- 508 standard cells
  - 504 C-Elements
  - 4 metastability filters
ASCEnD-ST65 (Current - 2019)

- Targets STM65 Bulk CMOS
- 128 cell types
  - Includes C-elements
  - Set/Reset/No control
  - Sutherland and van Berkel topologies
  - NCL/NCLP/INCL/INCLP gates
- MUTEX cells
- Several driving strengths → X2 to X31
- Total of 573 ready-to-use cells, 507 to go...
- All models available (LEF, Spice, Verilog, Liberty)
ASCEnD-FreePDK45

- **FreePDK45**: open access-based PDK for 45nm technology node → Predictive Technology Model (PTM)

- Open access asynchronous cell library with 30 cells

- Contains NCL/NCL+ gates → supports NCL/NCL+/SDDS designs

<table>
<thead>
<tr>
<th>Name</th>
<th>Family</th>
<th>Driving Strengths</th>
<th>Additional Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCL2W11OF2</td>
<td>NCL</td>
<td></td>
<td>SET - RESET</td>
</tr>
<tr>
<td>INCL2W11OF2</td>
<td>NCL</td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>NCL1W11OF2</td>
<td>NCL</td>
<td>X1, X2 and X4</td>
<td>NONE</td>
</tr>
<tr>
<td>INCL1W11OF2</td>
<td>NCL</td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>NCLP2W11OF2</td>
<td>NCLP</td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>INCLP2W11OF2</td>
<td>NCLP</td>
<td></td>
<td>NONE</td>
</tr>
</tbody>
</table>
An Example Cell – NCL2W11OF2
Ongoing Work on Flow/Libraries

• ASCEnD-TSMC180 → A library to allow fabricating asynchronous ICs using the Mini@sic Program of Europractice
  – Support for IoT edge nodes (Async!)
  – NCL / NCL+ / SDDS-NCL / Velo / SDDS-Velo

• Cooperation IHP-PUCRS → possibly, a cell library to further async design research in QDI
2. Models and Design Templates
Ordering of Models Presentation

• Bottom-up approach
  • Protocols first → with a bit of encoding
  • Communication links next → emphasis on data encoding
• Protocols + Comm. Links → channels + template families (BD and DI/QDI)
• Sets of Components → gate types and components distinct from sync design
• Architectures → async pipelines, flow specs
Asynchronous Protocols (4-phase and ST)

(Beerel et al., Async VLSI, Cambridge, 2010)

4-phase (BD) Protocol

Sender drives 1 of the N wires high

Receiver drives 1 of the N wires low

1 of the N wires rises (N-1 remains zero)

1-of-N

Data stable

4-phase (DI) Protocol

Single-Track Protocol

Sender drives 1 of the N wires high

Receiver drives 1 of the N wires low

(Beerel et al., Async VLSI, Cambridge, 2010)
Early, Late, Broad Protocol Versions

Narrow/Early:
- Data stable after Req+
- Data stable until Ack+

Late:
- Data stable after Req-
- Data stable until Ack-

Broad:
- Data stable after Req+
- Data stable until Ack-
• Two-phase Bundled-Data Protocol
  • Rising or falling transitions on Req → New data is available
  • Rising or falling transitions on Ack → Data was acknowledged
  • Sometimes called transition signalling
  • Transition is meaningful, not the signal level
The 1-of-N Nature of (Some) Protocols

1-of-2
4-phase
2 wires per bit

1-of-4
4-phase
4 wires \rightarrow 2 bits

More on this later on!!
Pull Protocols

4-phase (BD) Pull Protocol

Early:
- Data stable after Ack+
- Data stable until Req-

Late:
- Data stable after Ack-
- Data stable until Req+

Broad:
- Data stable after Ack+
- Data stable until Req-

Req/Ack reversed, Receiver starts handshake
Communication Links ➔ Data Encoding

• How to represent information in async circuits ➔ Options
  • Boolean or Single Rail ➔ as in digital sync circuits
    1 bit = 1 wire
  • Delay Insensitive (DI)
    – Basic idea ➔ merge data validity (control) and value!
  • Examples of DI codes
    – 1-of-2 ➔ 1 bit = 2 wires (special case of m-de-n)
    – m-of-n ➔ n wires = C_{m,n} distinct values representable
    – There are others (Berger codes, Knuth codes, etc.)
  • Often used codes: Dual Rail (DR) and 1-of-4

(See more on “Verhoeff, DI Codes, Distributed Computing, 1988.”)
The Dual Rail (DR) Concept

<table>
<thead>
<tr>
<th>a1a0</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>No data</td>
</tr>
<tr>
<td>01</td>
<td>‘0’ bit value</td>
</tr>
<tr>
<td>10</td>
<td>‘1’ bit value</td>
</tr>
<tr>
<td>11</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

Note:
- 1-of-2 represents 1 bit in 2 wires. 1-of-4 uses 4 wires to represent 2 bits
- However, 2x1-of-2 (a 2-bit DR) is a code different from 1-of-4!! Check it…
Examples of DI Codeword Sets

Dual Rail for 2 bits: 0101, 0110, 1001, 1010 (equivalent e.g. to Boolean 00, 01, 10, 11)
- The value 0000 is an invalid codeword that represents absence of data.
- All other 11 codewords of 4 binary digits are invalid and unused!

1-of-4: 0001, 0010, 0100, 1000 (equivalent e.g. to Boolean 00, 01, 10, 11)
- The value 0000 is an invalid codeword that represents absence of data.
- All other 11 codewords of 4 binary digits are invalid and unused!

3-of-5: 00111, 01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010, 11100
- The value 00000 is an invalid codeword that represents absence of data.
- All other 21 codewords of 4 binary digits are invalid and unused!

How many bits a 3-of-5 code can represent? 2,5?
When a communication link combines with a protocol, a channel is formed
- Comprises wires for carrying data (or not)
- Comprises wires for control (or not)

Channel nature defines the two major asynchronous design template families
- Bundled-Data (BD) $\rightarrow$ data encoded using binary codes, handshake considers data delay, adapts control signal timing
- DI/QDI $\rightarrow$ data encoded using some DI code, data validity detected at receiver

Channels can also be classified by whom initiates communication (slide right side)

- **Push Channel**
- **Pull Channel**

Handshaking details omitted
The Main Delay Models

- Gate delay model: delays in gates, no delays in wires

- Wire delay model: delays in gates and wires
An Async Taxonomy

- **Bounded delays (BD):** Realistic for gates and wires.
  - Technology mapping is easy, verification is difficult

- **Speed independent (SI):** Unbounded (pessimistic) delays for gates and “negligible” (optimistic) delays for wires.
  - Technology mapping is more difficult, verification is easy

- **Delay insensitive (DI):** Unbounded (pessimistic) delays for gates and wires.
  - DI class (built out of basic gates) is almost empty

- **Quasi-delay insensitive (QDI):** Delay insensitive except for critical wire forks (*isochronic forks*).
  - In practice it is the same as speed independent
Async Component Set

The most basic distinct async component is the C-element
- Simplest 2-input component
  → if inputs are equal, output is equal to these
  → if inputs differ, output keeps previous value

Truth Table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Qi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Qi-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Qi-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Some Transistor Topologies

(a) Martin
(b) Sutherland
(c) van Berkel

Useful to synchronize events!
NCL Gates

NCL gates
• threshold function + hysteresis
• Inputs can have weights \( w_i \)
• If sum of all \( w_i \cdot \text{inp}_i \) >= M, \( Q \rightarrow 1 \)
• If \( w_i \cdot \text{inp}_i = 0 \), \( Q \rightarrow 0 \)
• Otherwise \( Q(i) \rightarrow Q(i-1) \)

• A very different way to design digital circuits!

A basic subset of NCL gates \( \rightarrow \)
note the equivalency of some gates to OR gates and other to C-elements
Other Components

- **Mutexes:** Relevant for building arbiters

- **Domino Logic gates:** (pseudo-)Dynamic elements with multiple outputs used to build high performance asynchronous circuits

- **Asymmetric C-elements:** Useful in implementing multiple protocols

- **Etc, etc.**
Architectures and Design Styles

- Flow control Models → Petri Nets
  - Asynchronous State Transition Graphs (ASTGs)
  - Marked Graphs (MGs)
- Basic Asynchronous pipeline organizations
  - Interconnect ordinary and async components
  - Follow a template
  - Templates can be half buffer or full buffer
    - Half-buffer → one data every two stages
    - Full-buffer → one data at every stage
Petri Nets Representations

Primary Input

ASTG

Equivalent Petri Net
The Muller Pipeline (Sparso & Furber)

- "The" delay-insensitive handshake machine
- C[i] accepts 1/0 from C[i-1] only if C[i+1]=0/1
- Think of 1010101.. as waves: 10 10 10 1..
- C-elements propagate waves precisely
- Timing depends on local delays, may vary along the pipe
- If RIGHT is quiet, pipe will fill and stall
- Same for 4-phase, 2-phase
- Symmetric – same right-to-left (like electrons and holes)
Pipeline Styles

• Most based on Muller Pipeline
• 4-phase bundled data
  – similar to sync pipes
  – based on timing assumptions
• 2-phase bundled data
  – a.k.a. micropipelines
• 4-phase dual rail
  – “the original” Muller pipe
4-Phase BD Pipelines
4-Phase BD Pipelines

• Looks like a sync pipe, with local clocks
• When full, the C-elements are 1010101…,
  → only half the latches store data!
  → Accordingly, named a half-buffer scheme!
• Similar to master-slave flip-flops
• Speed limited by handshake (2-way comm)
• Transition signaling
• Special “capture-pass” latches alternate between capture and pass
Capture-Pass Transition-controlled Latch

- Transitions on C and P alternate
- Micropipelines → “Elegant”, no RTZ overhead
- But implementation (latches and other control circuits) is complex
4-Phase DR Pipelines

- Muller pipeline (again) with completion detection
- No REQ – embedded in data
4-Phase DR Pipelines – Many Bits

ACK

d[0].t

d[0].f

d[1].t

d[1].f

ACK

d[0].t

d[0].f

d[1].t

d[1].f
4. QDI Design - Some advances
Compatible with 4-phase DI channels

- **Alternative to RTZ channel – SBCCI12**
- **E.g. 1-of-2**

### RTZ

<table>
<thead>
<tr>
<th>Value</th>
<th>d.1</th>
<th>d.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacer</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Valid 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Valid 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Invalid</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### RTO

<table>
<thead>
<tr>
<th>Value</th>
<th>d.1</th>
<th>d.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Valid 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Valid 0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Spacer</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Return-to-One: Related Work

- For cryptographic cores design
  - Sokolov et al. [Sok06], [SMBY05]
    - Dual spacers
  - Cilio et al. [CLP+10]
    - Dual spacers
  - Murphy and Yakovlev [MY06]
    - Prototype on silicon
  - Moore et al. [MAM+03b]
    - Alarm codeword
Return-to-One: NCL+

- NCL Design (Fant and Brandt [FB96])
  - NCL gates
  - E.g. 3W211-of-3

\[
Q_i = \begin{cases} 
  1, & \sum_{j=0}^{n-1} w_j x_j \geq T \\
  0, & \sum_{j=0}^{n-1} w_j x_j = 0 \\
  Q_{i-1}, & 0 < \sum_{j=0}^{n-1} w_j x_j < T 
\end{cases}
\]

<table>
<thead>
<tr>
<th>$X_0$</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$Q_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$Q_{i-1}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$Q_{i-1}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$Q_{i-1}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$Q_{i-1}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Return-to-One: NCL+

- RTO version of NCL – MWSCAS13
  - NCL vs NCL+ cells: e.g. 2W111-of-3

\[
Q_i = \begin{cases} 
1, & \sum_{j=0}^{n-1} w_j x_j \geq T \\
0, & \sum_{j=0}^{n-1} w_j x_j = 0 \\
Q_{i-1}, & 0 < \sum_{i=n}^{n-1} w_j x_j < T 
\end{cases}
\]

\[
Q_i = \begin{cases} 
1, & \sum_{j=0}^{n-1} w_j \overline{x}_j = 0 \\
0, & \sum_{j=0}^{n-1} w_j \overline{x}_j \geq T \\
Q_{i-1}, & 0 < \sum_{i=n}^{n-1} w_j \overline{x}_j < T 
\end{cases}
\]
Return-to-One: NCL+

- Better power efficiency
SDDS-NCL: A New QDI Template

- NCL/NCL+ \(\rightarrow\) Spatially Distributed Dual-Spacer – JOLPE14
  - **Negative unate** gates
  - Better power, area and speed trade offs
  - Full **scope of existing synthesis/optimization** tools
Positive virtual functions pairs

- E.g. \( Q = A(B+C) \)
  - 3W211 and 2W211

\[
\exists \theta \in \Theta : ON(f) = ON(\theta) \\
\exists \phi \in \Phi : OFF(f) = OFF(\phi)
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>( Q_{VF} )</th>
<th>( Q_{NCL} )</th>
<th>( Q_{NCL+} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
SDDS-NCL: Logic Synthesis

Negative virtual functions pairs

- E.g. $Q = \neg(AB)$
  - $2W11$ and $1W11$

$$\exists \psi \in \Psi : \text{OFF}(g) = \text{OFF}(\psi)$$
$$\exists \nu \in \Upsilon : \text{ON}(g) = \text{ON}(\nu)$$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$Q_{VF}$</th>
<th>$Q_{NCL}$</th>
<th>$Q_{NCL+}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Input HDL must employ VFs only

- Design constraints
- Conventional tools

```verilog
module HA (A.1, A.0, B.1, B.0, C.1, C.0, S.1, S.0);
  input A.1, A.0, B.1, B.0;
  output C.1, C.0, S.1, S.0;
  wire C.1, C.0, S.1, S.0;
  assign S.1 = (A.1 & B.0) || (A.0 & B.1);
  assign S.0 = (A.1 & B.1) || (A.0 & B.0);
  assign C.1 = (A.1 & B.1);
  assign C.0 = (A.1 & B.0) || (A.0 & B.0) || (A.0 & B.1);
endmodule
```
SDDS-NCL: Logic Synthesis

- X-Netlist generation
The Fix X-Netlist algorithm

```plaintext
for each cell do
  if primary inputs are RTZ then
    if inversion polarity is even then
      cell is NCL
    else
      cell is NCL+
  else if primary inputs are RTO then
    if inversion polarity is even then
      cell is NCL+
    else
      cell is NCL
```

---

[Diagram showing the process of logic synthesis, starting with a 1-of-n QDI Circuit Description, followed by Logic Synthesis and Optimization, then Mapped X-Netlist, Fix X-Netlist, Mapped SDDS-NCL Netlist, Tune NCL, and final mapped SDDS-NCL Netlist.]
SDDS-NCL: Logic Synthesis

Optimize design
• Only for comb circuits
SDDS-NCL: Physical Synthesis
SDDS-NCL: Building Virtual Libraries

Unate Boolean Function

Neuizing et al. Algorithm

NCL TLF

ASCEEnD-A Design Flow

Generate VCell

NCL Virtual Library

NCL Physical Library

Modified Neuizing et al. Algorithm

NCL+ TLF

ASCEEnD-A Design Flow

Generate VCell

NCL+ Virtual Library

NCL+ Physical Library
SDDS-NCL: Virtual Libraries

- All VFs of up to 4 inputs
- Inverted VFs
- 56 VFs

<table>
<thead>
<tr>
<th>#in</th>
<th>VF</th>
<th>TLF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NCL</td>
</tr>
<tr>
<td>2</td>
<td>A + B</td>
<td>1W11</td>
</tr>
<tr>
<td></td>
<td>A · B</td>
<td>2W11</td>
</tr>
<tr>
<td>(A · (B + C)) + (B · C)</td>
<td>2W111</td>
<td>2W111</td>
</tr>
<tr>
<td></td>
<td>A · (B + C)</td>
<td>3W211</td>
</tr>
<tr>
<td></td>
<td>A + B + C</td>
<td>1W111</td>
</tr>
<tr>
<td></td>
<td>A + (B · C)</td>
<td>2W211</td>
</tr>
<tr>
<td></td>
<td>A · B · C</td>
<td>3W111</td>
</tr>
<tr>
<td>3</td>
<td>(A · (B + C)) + (B · (C + D))</td>
<td>2W1111</td>
</tr>
<tr>
<td>(A · (B + C)) + (B · (C + D))</td>
<td>3W2211</td>
<td>4W2211</td>
</tr>
<tr>
<td>(A · (B + C)) + (B · C · D)</td>
<td>3W2111</td>
<td>3W2111</td>
</tr>
<tr>
<td>(A · (B + C)) + (B · C)</td>
<td>4W3221</td>
<td>5W3221</td>
</tr>
<tr>
<td>A · (B + C + D)</td>
<td>4W3111</td>
<td>3W3111</td>
</tr>
<tr>
<td>(A · (B + C)) + (B · (C + D))</td>
<td>4W2321</td>
<td>5W2321</td>
</tr>
<tr>
<td>(A · (B + C)) + (B · C · D)</td>
<td>5W3221</td>
<td>4W3221</td>
</tr>
<tr>
<td>(A · B) + (A · C · D) + (B · C · D)</td>
<td>4W2211</td>
<td>3W2211</td>
</tr>
<tr>
<td>(A · B) + (A · C · D)</td>
<td>5W3211</td>
<td>3W3211</td>
</tr>
<tr>
<td>(A · B · (C + D)) + (A · C · D) + (B · C · D)</td>
<td>3W1111</td>
<td>2W1111</td>
</tr>
<tr>
<td>(A · B · C) + (A · B · D) + (A · C · D)</td>
<td>4W2111</td>
<td>2W2111</td>
</tr>
<tr>
<td>A · B · (C + D)</td>
<td>5W2211</td>
<td>2W2111</td>
</tr>
<tr>
<td>A + (B · (C + D)) + (C · D)</td>
<td>2W2111</td>
<td>4W2111</td>
</tr>
<tr>
<td>A + (B · (C + D))</td>
<td>3W3211</td>
<td>5W3211</td>
</tr>
<tr>
<td>A + B + C + D</td>
<td>1W1111</td>
<td>4W1111</td>
</tr>
<tr>
<td>A + B + (C · D)</td>
<td>2W2211</td>
<td>5W2211</td>
</tr>
<tr>
<td>A + (B · C · D)</td>
<td>3W3111</td>
<td>4W3111</td>
</tr>
<tr>
<td>A · B · C · D</td>
<td>4W1111</td>
<td>1W1111</td>
</tr>
<tr>
<td>(A · B) + (A · C) + (B · D)</td>
<td>Special0</td>
<td>Special0</td>
</tr>
<tr>
<td>(A · B) + (C · D)</td>
<td>Special1</td>
<td>Special1</td>
</tr>
<tr>
<td>(A · C) + (A · D) + (B · C) + (B · D)</td>
<td>Special2</td>
<td>Special2</td>
</tr>
</tbody>
</table>
**SDDS-NCL: Design Space Exploration**

- First case studies – **ASYNC14**
  - All combinational
  - Different degrees of complexity
  - Two synthesis modes
  - Exploration of constraints and STA

<table>
<thead>
<tr>
<th>Gates</th>
<th>8bRC</th>
<th>8bKS</th>
<th>32bKS</th>
<th>32bALU</th>
<th>16bMUL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relaxed</td>
<td>Fast</td>
<td>Relaxed</td>
<td>Fast</td>
<td>Relaxed</td>
</tr>
<tr>
<td>Total</td>
<td>247</td>
<td>282</td>
<td>422</td>
<td>521</td>
<td>2942</td>
</tr>
<tr>
<td>% NCL</td>
<td>41</td>
<td>35</td>
<td>49</td>
<td>41</td>
<td>48</td>
</tr>
<tr>
<td>% NCL+</td>
<td>27</td>
<td>28</td>
<td>27</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>% INV</td>
<td>32</td>
<td>37</td>
<td>24</td>
<td>35</td>
<td>26</td>
</tr>
<tr>
<td>Crit. Path</td>
<td>60</td>
<td>48</td>
<td>24</td>
<td>18</td>
<td>36</td>
</tr>
<tr>
<td>Latency (ns)</td>
<td>Forward</td>
<td>3.90</td>
<td>1.41</td>
<td>2.33</td>
<td>1.55</td>
</tr>
<tr>
<td></td>
<td>Transmission</td>
<td>7.71</td>
<td>2.72</td>
<td>4.41</td>
<td>3.07</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>Idle</td>
<td>0.014</td>
<td>0.021</td>
<td>0.030</td>
<td>0.041</td>
</tr>
</tbody>
</table>
SDDS-NCL vs NCL

- 16 bit multipliers
  - Maximum performance
  - Up to 4 inputs for
  - template-based
  - 614 cells → 56 VFs

<table>
<thead>
<tr>
<th></th>
<th>NCL</th>
<th>SDDS-NCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (MOPS)</td>
<td>103.50</td>
<td>277.78</td>
</tr>
<tr>
<td>Leakage Power (mW)</td>
<td>0.124</td>
<td>0.137</td>
</tr>
<tr>
<td>Dynamic Power (mW)</td>
<td>0.478</td>
<td>0.739</td>
</tr>
<tr>
<td>Total Power (mW)</td>
<td>0.602</td>
<td>0.876</td>
</tr>
<tr>
<td>Gate Count</td>
<td>3336</td>
<td>3963</td>
</tr>
<tr>
<td>Area mm²</td>
<td>0.0290</td>
<td>0.0197</td>
</tr>
<tr>
<td>Normalized MOPS / Leak. Power</td>
<td>1</td>
<td>2.428</td>
</tr>
<tr>
<td>Normalized MOPS / Dyn. Power</td>
<td>1</td>
<td>1.737</td>
</tr>
<tr>
<td>Normalized MOPS / Total Power</td>
<td>1</td>
<td>1.845</td>
</tr>
<tr>
<td>Normalized MOPS / Area</td>
<td>1</td>
<td>3.940</td>
</tr>
</tbody>
</table>
ASYNC’19 Paper Presentation

“Pulsar - Constraining QDI Circuits Cycle Time Using Traditional EDA Tools”
Marcos Luiggi Lemos Sartori
Rodrigo Nogueira Wuerdig
Matheus Trevisan Moreira
Ney Laert Vilar Calazans

BPA Nominee
5. Async Analysis – Test of asynchronous circuits
Challenges

• VLSI test $\rightarrow$ requirement for any commercializable circuit
  • Including asynchronous circuits

• There are several different asynchronous templates
  • Every template $\rightarrow$ a distinct test approach
  • No DfT automation available
  • Synchronous DfT techniques $\rightarrow$ do not work out-of-the box
Challenges

• Compared to synchronous sequential circuits
  • No global clock → lower controllability
  • Timing nondeterminism
    • No cycle accurate response expectable by testers
  • Function nondeterminism
    • Arbitration with C-element can lead to nondeterminism
    • No way to predict expected response

• Asynchronous usu. → redundant logic
  • Faults in redundant logic
    → do not affect the response
    → can create undetectable faults
Challenges

• How to add scan chains without a global clock?
  • One Solution → Add a clock just for test mode
• Async → typically use latches, not FFs
  • Use a scannable latch → LSSD with more overhead than mux’-D cell
• Async has feedback loops
  • In test mode → break loops with a scan cell

• Result → silicon area overhead for DfT >> synchronous circuits
  • Synchronous DfT overhead: from 5% to 12%
  • Asynchronous DfT overhead: around 25%
ASYNC Test Advantages

• Delay fault tolerance
  • delay faults in a DI circuit
    → typically fully tolerated
    → typically minor performance penalty

• Self-checking capability
  • Faults in control (handshake) corrupt the entire circuit
  • Faults → Easy to observe
Approaches - No Scan

- Logic between pipeline stages → tested as one logic block
  - All latches in pass mode
  - ATPG for combinational logic
- Control part halts for all stuck-at faults
- Delay faults only for the whole block, not for individual stages

* Issues in fault modeling and testing of micropipelines, by S. Pagey, ATS (1992)

Figure 2: Micropipeline with Processing Logic
No Control Halt

MouseTrap → does not halt for every stuck-at fault in control
- Also considers single block approach → testing stuck at faults in the combinational logic
- Some argue → time for generating the test patterns can be large
  - Full scan or partial scan may be better for long pipelines

• Development of custom scan cell
  • Lack support by commercial EDA tools
• Scan test control logic (STCL) \(\rightarrow\) adds asynchronous interface for scan control
  • Only active during test

• Built-in self-test (BIST) micropipeline design → based on an Async BILBO (Built-In Logic Block Observer) register
  • Random pattern generation and signature analysis intrachip
  • Area overhead may be prohibitive

NCL Partial Scan

- Break feedback loops → new NCL scan cell
- 100% of stuck-at fault coverage; ATPG with Tetramax
- Area overhead → 9% to 23%

Scan C-Element

- C-Elements have **combinational loops**
  - Create sequential behavior (storage effect)
- Loops must be broken for ATPG
- Conventional approach
  - add an LSSD scan cell
  - High area overhead

Scan C-Element

- Approach to reduce area
- Try sharing existing resources
- Latch $\rightarrow$ placed in parallel with loops
  - shift: $si \rightarrow so$
  - capture: $c \rightarrow d$

Scan C-Element

Results with the mux-based method

<table>
<thead>
<tr>
<th>C-element</th>
<th>Area (nand equiv.)</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No scan</td>
<td>LSSD</td>
</tr>
<tr>
<td>asym. 2 input</td>
<td>2</td>
<td>10.3</td>
</tr>
<tr>
<td>sym. 2 input</td>
<td>3</td>
<td>10.3</td>
</tr>
</tbody>
</table>

Table 2. Cell area of scan C-elements.

<table>
<thead>
<tr>
<th>C-element</th>
<th>Delay (ns.)</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No scan</td>
<td>LSSD</td>
</tr>
<tr>
<td>asym. 2 input</td>
<td>0.14</td>
<td>0.43</td>
</tr>
<tr>
<td>sym. 2 input</td>
<td>0.18</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Table 3. Delay of scan C-elements.

Results obtained with the mux-based method

<table>
<thead>
<tr>
<th>Circuit</th>
<th># C-elements</th>
<th># self loops</th>
<th>Non scan Area</th>
<th>LSSD Scan Area</th>
<th>Mux scan Area</th>
<th>Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80c51 μC</td>
<td>139</td>
<td>41</td>
<td>310</td>
<td>1352</td>
<td>783</td>
<td>42</td>
</tr>
<tr>
<td>ddd</td>
<td>537</td>
<td>85</td>
<td>1203</td>
<td>5743</td>
<td>2634</td>
<td>54</td>
</tr>
<tr>
<td>des</td>
<td>110</td>
<td>18</td>
<td>229</td>
<td>1088</td>
<td>582</td>
<td>46</td>
</tr>
<tr>
<td>sc</td>
<td>539</td>
<td>235</td>
<td>1172</td>
<td>5238</td>
<td>3401</td>
<td>35</td>
</tr>
<tr>
<td>adpcm</td>
<td>571</td>
<td>31</td>
<td>1232</td>
<td>5580</td>
<td>2636</td>
<td>52</td>
</tr>
</tbody>
</table>

Table 4. Area of the C-elements in the benchmark circuits