Advances in Designing Clockless Digital Systems

Prof. Steven M. Nowick

nowick@cs.columbia.edu

Department of Computer Science (and Elect. Eng.)
Columbia University
New York, NY, USA
Introduction

- **Synchronous vs. Asynchronous Systems?**
  - **Synchronous Systems**: use a *global clock*
    - entire system operates *at fixed-rate*
    - uses “*centralized control*”

![Diagram](image-url)
Synchronous vs. Asynchronous Systems? (cont.)

- **Asynchronous Systems**: *no global clock*
  - components can operate at *varying rates*
  - *communicate locally via “handshaking”*
  - uses *“distributed control”*

“handshaking interfaces” (channels)
Trends and Challenges

Trends in Chip Design: next decade

- “Semiconductor Industry Association (ITRS) Roadmap”

Unprecedented Challenges:

- complexity and scale (= size of systems)
- clock speeds
- power management
- reusability & scalability
- reliability
- “time-to-market”

Design becoming unmanageable using a centralized single clock (synchronous) approach....
1. Clock Rate:

* 1980: several MegaHertz
* 2013: 2-6 GigaHertz (and falling!)

Design Challenge:

* “clock skew”: clock must be near-simultaneous across entire chip
2. Chip Size and Density:

Total #Transistors per Chip: 60-80% increase/year

- 1969: 4 thousand (Intel 4004 microprocessor)
- 2013 and beyond: 1-3 billion+

Design Challenges:

- system complexity, design time, clock distribution
- clock will require 10-20 cycles to reach across chip
3. Power Consumption

- Low power: ever-increasing demand
  - *consumer electronics*: battery-powered
  - *high-end processors*: avoid expensive fans, packaging

Design Challenge:

- *clock* *inherently consumes power continuously*

- “power-down” techniques: add complexity, only partly effective
4. Time-to-Market, Design Re-Use, Scalability

Increasing pressure for faster “time-to-market”. Need:

- **reusable components**: “plug-and-play” design
- **flexible interfacing**: under varied conditions, voltage scaling
- **scalable design**: easy system upgrades

Design Challenge: mismatch with central fixed-rate clock
5. Future Trends: “Mixed Timing” Domains

Chips themselves becoming distributed systems....

* contain many sub-regions, operating at different speeds:

Design Challenge: breakdown of single centralized clock control
Asynchronous Design: Potential Advantages

Lower Power

- **no clock**
  - components inherently use dynamic power only “on demand”
  - **no global clock distribution**
  - effectively provides automatic clock gating at arbitrary granularity

Robustness, Scalability

- **no global timing**
  - “mix-and-match” variable-speed components
  - supports dynamic voltage scaling
  - modular design style ➔ “object-oriented”

Higher Performance (**sometimes**)

- not limited to “worst-case” clock rate

“Demand- (Data-) Driven” Operation

- instantaneous wake-up from standby mode
Example: Current Comparison – 80c51 Microcontroller
Asynchronous Design: Potential Targets

Large variety of asynchronous design styles

* Address different points in “design-space” spectrum...
  
  * **extreme timing-robustness:**
    * providing near “delay-insensitive (DI)” operation
  
  * **ultra-low power, energy:**
    * “on-demand” operation, instant wakeup
  
  * **ease-of-design/moderate performance/low EMI** (electro-magnetic interference)
    * e.g. goal at Philips Semiconductors
  
  * **very high-speed:** asynchronous pipelines
    * ... comparable to high-end synchronous
    * with added benefits: support variable-speed I/O rates
  
  * **support for heterogeneous systems:** integrate clock domains via async
    * “GALS-style” (globally-async/locally-sync)
  
  * **use in emerging technologies:** QCA, CNT, photonic/digital, etc.
Asynchronous Design: Recent Industrial Developments

**Intel** (2011, acquired Fulcrum Microsystems startup): Ethernet routing chips
  *
  now Intel’s “Switch and Routing Division (SRD)” = **fully-asynchronous**

**Achronix Semiconductor**: *high-speed FPGA’s* -- "world-leading" *(1.5 GHz)*
  *
  2011: Intel to fab Speedster 22i their 22nm technology *(1st such agreement)*

**Philips Semiconductors (to late 1990’s)**: > 700 million microcontrollers
  *
  for smartcards, e-passports, automotive, cell phones, etc.
Components usually communicate & synchronize on **channels**

![Diagram](image)

- **Sender**
- **Receiver**
- **channel**
- **without data**
**Overview: Signalling Protocols**

*Communication channel:* usually instantiated as 2 wires

*Sender*  

*Receiver*

*without data*
Overview: Signalling Protocols

4-Phase Handshaking

One transaction (return-to-zero [RZ]):

Active (evaluate) phase

Return-to-zero (RZ) phase
Overview: Signalling Protocols

2-Phase Handshaking = “Transition-Signalling”

Two transactions (non-return-to-zero [NRZ]):
**Data channel:** replace “req” by (encoded) data bits
- ... still use 2-phase or 4-phase protocol
Overview: How to Encode Data?

A variety of asynchronous data encoding styles

* Two key classes: (i) “DI” (delay-insensitive) or (ii) “timing-dependent”
* ... each can use either a 2-phase or 4-phase protocol

**DI Codes:** provides timing-robustness ➔ to arbitrary bit skew, input arrival time, etc.

* 4-phase (RZ) protocols:
  * dual-rail (1-of-2): *widely used!*
  * 1-of-4 (or m-of-n)

* 2-phase (NRZ) protocols:
  * transition-signaling (1-of-2)
  * LEDR (1-of-2) [“level-encoded dual-rail”] [Dean/Horowitz/Dill, Adv. Research in VLSI ’91]
  * LETS (1-of-4) [“level-encoded transition-signalling”] [McGee/Agyekum/Mohamed/Nowick IEEE Async Symp. ’08]
Overview: How to Encode Data?

“dual-rail”: 4-Phase (RZ)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Dual-rail encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X1  X0</td>
</tr>
<tr>
<td>0</td>
<td>0 1</td>
</tr>
<tr>
<td>1</td>
<td>1 0</td>
</tr>
<tr>
<td>no data</td>
<td>0 0 = NULL (spacer)</td>
</tr>
</tbody>
</table>

Sender

Receiver

ack
### Overview: How to Encode Data?

#### “1-of-4”: 4-Phase (RZ)

<table>
<thead>
<tr>
<th>Bits A B</th>
<th>Dual-rail encoding X3 X2 X1 X0</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>01</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>10</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>11</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>no data</td>
<td>0 0 0 0 = NULL (spacer)</td>
</tr>
</tbody>
</table>

Sender

Receiver
Overview: How to Encode Data?

Single-Rail “Bundled Data”: 4-Phase (RZ)

Uses synchronous single-rail data (potentially glitchy!) + local worst-case delay

“bundling” signal

Sender

Receiver

req

A

B

ack
Critical Design Issues:

- Components must communicate cleanly: ‘hazard-free’ design
- Highly-concurrent designs: harder to verify!

Lack of Automated “Computer-Aided Design” Tools:

- Most commercial “CAD” tools targeted to synchronous
Asynchronous Design Challenge

Lack of Existing Asynchronous Design Tools:

- Most commercial “CAD” tools targeted to synchronous

- Synchronous CAD tools:
  - major drivers of growth in microelectronics industry

- Asynchronous “chicken-and-egg” problem:
  - few CAD tools ↔ less commercial use of async design
  - especially lacking: tools for designing/optmzng. large systems
CAD Tools for Async Controllers

MINIMALIST: developed at Columbia University [1994-]
- extensible CAD package for synthesis of asynchronous controllers
- integrates synthesis, optimization and verification tools
- used in 80+ sites/17+ countries (was taught in IIT Bombay)
- URL: http://www.cs.columbia.edu/~nowick/asynctools

Features:
- Automatic design scripts + custom commands
- Performance-driven multi-level logic decomposition
- Verilog back-end
- Automatic verifier
- Graphical interfaces
- ... many optimization modes

Recent application: laser space measurement chip (joint with NASA Goddard)
- NASA/Columbia (2006-2007)
- experimental chip: taped out (Oct. 06)

Key goal: facilitate design-space exploration
Example: “PE-SEND-IFC” (HP Labs)

**Inputs:**
- req-send
- treq
- rd-iq
- adbld-out
- ack-pkt

**Outputs:**
- tack
- peack
- adbld

From HP Labs
“Mayfly” Project:
B. Coates, A. Davis, K. Stevens,
“The Post Office Experience: Designing a Large Asynchronous Chip”,
EXAMPLE (cont.):
Design-Space Exploration using MINIMALIST: optimizing for area vs. speed
NON-PIPELINED COMPUTATION:

“datapath component” = adder, multiplier, etc.

global clock

SYNCHRONOUS
“PIELED COMPUTATION”: like an assembly line

**SYNCHRONOUS**

**ASYNCHRONOUS**
MOUSETRAP: A Basic FIFO (no computation)

Stages communicate using *transition-signaling (2-phase)*:

![Diagram of stages communicating using transition-signaling](image)

Features: standard cell design, single D-latch register per stage

[Singh/Nowick, IEEE Trans. on VLSI Systems (June 2007), ICCD (2001)]
“MOUSETRAP” Pipeline: adding computation

Function Blocks: use “synchronous” logic blocks (not hazard-free!) 
+ a local “matched delay” \((req)\)

“Bundled Data” Requirement \((1\text{-sided})\):
- Each \(req\) must arrive after data inputs valid and stable
High-Speed Asynchronous Pipelines

Goal: fast + flexible async datapath components

- **speed:** comparable to fastest existing synchronous designs
- **additional benefits:**
  - dynamically adapt to variable-speed interfaces/handle DVFS
  - no requirement of balanced stages/avoid clock distribution

Contributions: 3 New Asynchronous Pipeline Styles (Singh/Nowick)

(i) MOUSETRAP: static logic  [ICCD-01, IEEE Trans. on VLSI Systems 2007]
(ii) Lookahead (LP): dynamic logic  [Async-02, IEEE Trans. on VLSI Systems 2007]
(iii) High-Capacity (HC): dynamic logic  [Async-02, ISSCC-02, IEEE Trans. on VLSI Systems 2007]


- async filter in sync wrapper
- provides “adaptive latency” = # of clock cycles per operation
- performance: better than leading comparable IBM synchronous design
**Mixed-Timing Interfaces: Challenge**

**Goal:** provide low-latency communication between "timing domains"

**Challenge:** avoid synchronization errors
Mixed-Timing Interfaces: Solution

Solution: insert mixed-timing FIFO’s ⇒ provide safe data transfer
... developed complete family of mixed-timing interface circuits
Our approach widely used, in architecture and NOC communities
- esp. for multi-clock systems, dynamic voltage/frequency scaling (DVFS)

Examples:

* **Architecture:** A. Gonzales/G. Magklis (Intel-Barcelona),
  D. Albonesi (Cornell), D. Marculescu (CMU), M. Martonosi (Princeton)

* **NOC:** J. Flich (U. Valencia), R. Marculescu (CMU), (Cornell),
  P. Vivet/E. Beigne (CEA-LETI: variant)
1. GALS Networks-on-Chip (NoC): for Shared-Memory Parallel Processors
   * Medium-scale NSF project [2008-12]: with Prof. Uzi Vishkin (University of Maryland)
   * Goal: low-power/high-performance async routing network (processors <=> memory)
     * “GALS”-style: globally-asynchronous/locally-synchronous
   * [M. Horak, S.M. Nowick, M. Carlberg, U. Vishkin, NOCS-10]

2. Continuous-Time DSP’s
   * Medium-scale NSF project [2010-14]: with Prof. Yannis Tsividis (Columbia EE Dept.)
   * Idea: adaptive signal processing, based on signal rate-of-change
   * Goal: low-aliasing + low-power -- combine analog + async digital

3. Asynchronous Bus Encoding: for Timing-Robust Global Communication
   * Goal: low-power, error-correction + timing-robust (“delay-insensitive”) communication
   * [M. Agyekum/S.M. Nowick, DATE-10, IWLS-10]

4. Variable-Latency Functional Units: “Speculative Completion”
   * Goal: high-performance components with ‘data-dependent’ completion
   * [S.M. Nowick et al., IEE Proceedings ‘96; IEEE Async-97 Symposium]
Some of My Recent Projects

1. Async/GALS Interconnection Networks: Shared-Memory Multicore Systems
   - Medium-scale NSF project [2008-13]: with Uzi Vishkin (U. Maryland)
   - Goal: low-power/high-performance async routing network (processors <-> memory)
     - "GALS"-style: globally-asynchronous/locally-synchronous
     - [M. Horak, G. Gill, S.M. Nowick, et al., ACM NOCS-10/NOCS-11 Symposium]

2. Continuous-Time (CT) DSP’s:
   - Medium-scale NSF project [2010-14]: with Prof. Yannis Tsividis (Columbia EE Dept.)
   - Idea: adaptive signal processing, based on signal rate-of-change
   - Goal: low-aliasing + low-power – combines real-time + async digital

3. Asynchronous Bus Encoding: for Timing-Robust Global Communication
   - Goal: low-power, error-correction + “delay-insensitive” communication
     - [M. Agyekum/S.M. Nowick, DATE-10, DATE-11, TCAD-12]

4. Ultra-Low Voltage (ULV) Digital Systems: through asynchrony
   - Collaboration with Prof. Mingoo Seok (Columbia EE Dept.)
   - Goal: target reliable and ultra-low-energy sensors, medical implants
Goal: fast analytical techniques + tools
- to handle large/complex asynchronous + mixed-timing systems
  * using stochastic delay models (Markovian): [P. McGee/S.M. Nowick, CODES-05]
  * using bounded delay models (min/max): [P. McGee/S.M. Nowick, ICCAD-07]

Applications: analysis + optimization

* Large Asynchronous Systems:
  * Evaluate latency, throughput, critical vs. slack paths, average-case performance
  * Drive optimization: pipeline granularity, module selection

* Large Heterogeneous (mixed-clock) or “GALS” Systems:
  * Evaluate critical vs. slack paths
  * Drive optimization: dynamic voltage scaling, load balancing of threads, buffer insertion
MLO is an integrated **post-processing** (i.e. backend) tool for Minimalist.

Targeted to **multi-level logic**.
- In contrast, Minimalist currently is targeted to two-level logic.

Designed to work on **combinational hazard-free logic** for Burst Mode controllers.
- Uses “hazard-non-increasing” transforms.

Output of MLO is Verilog.

MLO is a standalone tool running from the Linux shell **outside of Minimalist**.
Minimalist: MLO (Multi-Level Optimizer)

- Accessible on the web from:
  - Initial Release
    - One version – for Linux Distributions


- Includes
  - Complete Tutorial
  - Documentation
  - Examples

- Tool requires Python interpreter to run:

- Consult README for MLO installation information

CEO Feature - User-Specified Critical Events

User-Specified Critical Arcs Highlighted in **Red**

### Case 1: Non-colorized arc. User-Specified nothing is critical. Defaults to automated mode for every output.

### Case 2: Some outputs colorized, some outputs not. Both user-specified data and automated approaches are used to determine criticality. ITEventReq will use user-specified data to determine criticality. CtrincReq will default to automated mode to determine criticality.

### Case 3: Every output is colorized. Automated approach is never used. IntITReq- is critical with respect to CtrIncReq-, while ITEvent2Ticks- is NOT critical to CtrIncReq-
The next four slides present different MLO output examples. For each example, the starting circuit (input to MLO) is this circuit:

Two-level Structure from Minimalist Output
Feature Set Example 1 - Gate Fan-in Limitation

Result of MLO: Multi-Level circuit with AND gate fan-in limit of 2
Feature Set Example 2 - Negative Logic

This mode carefully optimizes only hazard non-increasing safe transformations (DeMorgan’s Law). Optimizations are also included to carefully eliminate extra inverters.
Feature Set Example 3 - CEO
“critical event optimizer”

Result of MLO: Multi-Level Circuit after MLO CEO is used

Gate Decomposed. Input intitreq is more critical to output iteventreq than ctrincack’ and y0’

critical primary input-to-output path
Result of MLO: Multi-Level Circuit with negative logic, AND gate fan-in limit of 2, and CEO.
MOUSERTRAP: A Basic FIFO

Stages communicate using *transition-signaling*:

1 transition per data item!

One Data Item
What Are CAD Tools?

Software programs to aid digital designers = 
"computer-aided design" tools

* automatically \textit{synthesize} and \textit{optimize} digital circuits

\textbf{Input:} desired circuit specification

\textbf{Output:} optimized circuit implementation