Asynchronous Circuits and Systems

Ken Stevens
University of Utah

Asynchronous Circuits and Systems

All our designs operate in a physical two-dimensional space on a wafer.

Timing has been referred to as *the third dimension*.

There is an overabundance of approaches to timing in our circuits.

Various Definitions of *Asynchronous* or *Self-Timed* design:

1. *A sequential system that operates without a fixed frequency clock*
2. *Sequential system that is reactive to environment*
3. *A system without centralized control over sequencing*
4. *Sequencing of design is specified without timing.*
5. *A system where sequencing and timing are connected in the interior of parts called elements*

My Biases and Perspective

- Significant Design Experience
  - Bit-sliced μ-sequencer: Soft controller
  - Network communication: Post Office
  - Pentium Processor front end: RAPPID
- Required 10x improvement over existing (crocked) design
  - 3×3: Performance and power
- Interested in high end
  - not where you want to introduce new technology
- The Big 3: Modularity, Power, Performance

Key Messages

- Asynchronous logic design techniques will be increasingly important to industry for:
  - power (thermal limits, batter life for mobile/wireless)
  - design reuse (high integration, PVT variations)
- Developing CAD for async will be key to applying the techniques effectively to our circuits
  - sequential design has different CAD requirements
  - state of the art is significantly behind

Most Often Cited Advantages

1.
2.
3.
4.
5.
6.
7.
8.
9.
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2.
3.
4.
5.
6.
7.
8.
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2.
3.
4.
5.
6.
7.
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2.
3.
4.
5.
6.
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2.
3.
4.
5.
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2.
3.
4.
5. Metastability has time to end
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2.
3.
4. No clock alignment at the interfaces
5. Metastability has time to end
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2.
3. Ease of modular composition
4. No clock alignment at the interfaces
5. Metastability has time to end
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1.
2. Power consumed only where needed
3. Ease of modular composition
4. No clock alignment at the interfaces
5. Metastability has time to end
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Most Often Cited Advantages

1. Achieve average case performance
2. Power consumed only where needed
3. Ease of modular composition
4. No clock alignment at the interfaces
5. Metastability has time to end
6. Avoid clock distribution costs
7. Easier to exploit concurrency
8. Intellectual challenge
9. Intrinsic elegance
10. Global synchrony doesn't exist anyway

Real Reasons - NOT Often Cited

1.
2.
3.
4.
5.
6.
7.
8.
9.
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2.
3.
4.
5.
6.
7.
8.
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2.
3.
4.
5.
6.
7.
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2.
3.
4.
5.
6.
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2.
3.
4.
5.
6. I don't understand synchronous circuits
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2.
3.
4.
5. People and circuits need to play by the same rules
6. I don't understand synchronous circuits
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2.
3.
4. Gee - I really don't know
5. People and circuits need to play by the same rules
6. I don't understand synchronous circuits
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2.
3. I like to be different
4. Gee - I really don't know
5. People and circuits need to play by the same rules
6. I don't understand synchronous circuits
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1.
2. I like reinventing wheels
3. I like to be different
4. Gee - I really don't know
5. People and circuits need to play by the same rules
6. I don't understand synchronous circuits
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

Real Reasons - NOT Often Cited

1. It really pisses my boss off
2. I like reinventing wheels
3. I like to be different
4. Gee - I really don't know
5. People and circuits need to play by the same rules
6. I don't understand synchronous circuits
7. World problems stem from glitches
8. Synchronous design gives me gas
9. Clock radiation causes hair loss
10. It's none of your business

What are the Right Questions?

1. Asynchronous design is absolutely discouraged in most industrial design houses.
2. Why?

What are the Right Questions?

1. Asynchronous design is absolutely discouraged in most industrial design houses.
2. Why?
   Races, CAD Tools, training, and the whole plethora . . .
3. Is this a correct perspective?

What are the Right Questions?

1. Asynchronous design is absolutely discouraged in most industrial design houses.
2. Why?
   Races, CAD Tools, training, and the whole plethora . . .
3. Is this a correct perspective?
   probably true, so why are we talking about this??!??!

What are the Right Questions?

1. Asynchronous design is absolutely discouraged in most industrial design houses.
2. Why?
   Races, CAD Tools, training, and the whole plethora . . .
3. Is this a correct perspective?
   probably true, so why are we talking about this??!??!
   . . . because we can address these issues and still have advantage.

What are *your* questions about async design?

What You should NOT Ask/State. . .

- Async designs are not { reliable | predictable | repeatable }
- Async designs have too much variation to interface to clocked design
- Async designs can not be made commercially viable

. . . because these are myths or *urban legends*

What are the Right Questions?

- What *is* async design?
- Why does async have a bad reputation?
- Are reactive designs a good thing?
- Is asynchronous design really faster?
- Is asynchronous design lower power?
- Where do theoretical advantages come from?
- What is relationship of Async and PVT variations?
- What are the overheads for handshaking?
- How does one achieve an advantage with this stuff?
- Does “wine-goblet” model apply to async design?

What are the Right Questions?

- Can Async circuits be tested, DFT?
- Is Async modularity for real?
- Is async modularity practical (IP Reuse)?
- What would best be designed async?
- How to interface to clocked design?

What are the Right Questions?

- What are typical protocols?
- Various protocol benefits and disadvantages?
- What is the relationship between timing and async circuits and protocols?
- Are races versus speed paths a big issue?
- What is key requirement for asynchronous protocol design?
- There are a million asynchronous design methodologies – how do they relate and which do I choose?

What are the Right Questions?

- What classes of circuits should I use?
- Do I need custom cell libraries?
- Why would one want to adopt async design? (The Big 3)
- Why would one not want to do async?
- Should I do async?

System Sequencing and Timing

	Sequencing and timing	Sequencing only
Centralized references	Traditional clocked design *Pros: Easy to validate, CAD* *Cons: Clock Power* *Best use: homogeneous designs*	Not useful
Local references	Timing ref travels with data *Pros: low latency, power efficient* *Cons: CAD support* *Best use: heterogeneous designs*	Traditional Async design *Pros: Easy to build basic system, robust* *Cons: Slow, power inefficient* *Best use:* designs with high variability

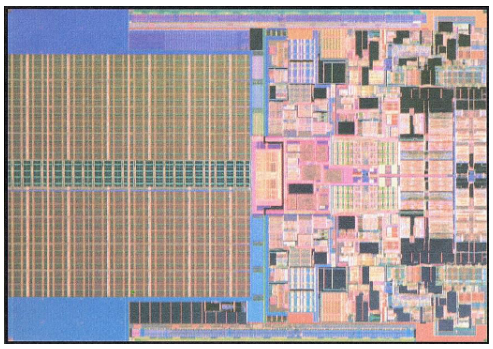
Clocked — sequencing derived from time (clock)

Self-timed — sequencing derived from data validity

"In the self-timed discipline, sequence and time are connected in the interior of parts called elements." Mead & Conway, 1980, p218

Crocked Tradition

However, design currently maintains a strong clocked strangle-hold



Intel 45nm Penryn Die Photo, Intel

Asynchronous Components Today

How much of our design today is asynchronous?

Asynchronous Components Today

How much of our design today is asynchronous?

- **Well over 50%!**

Almost all of this is due to the following two components:

Asynchronous Components Today

How much of our design today is asynchronous?

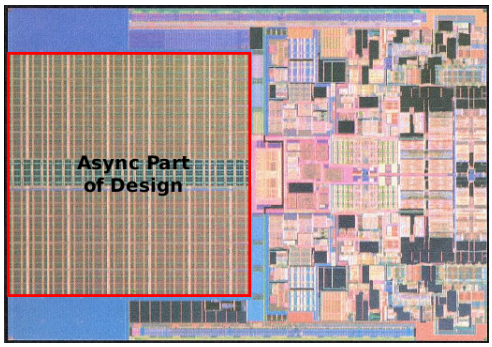
- **Well over 50%!**

Almost all of this is due to the following two components:

- RAM
  - no clock to the memory core
  - accessed with “req”
  - dual-rail data encodes “ack”
- Flops and latches
  - sequential asynchronous finite state machines
  - require correct ordering (relative timing) of data and clock

Crocked Tradition???

In reality, asynchronous design already covers ≈ half the die!



Intel 45nm Penryn Die Photo, Intel

History of Async Design

Much of the history of asynchronous design comes back to it’s Utah ties. . .

Async Conference started here

Just a few of the interesting players:

- Ivan Sutherland
  - Turing award recipient
  - Turing paper on “Micropipelines”
- Chuck Seitz
  - Started first successful Async company Myracom

Utah’s Historical Impact: IEEE Historical Society

CS & CE department spun out from EE: Dave Evans 1965

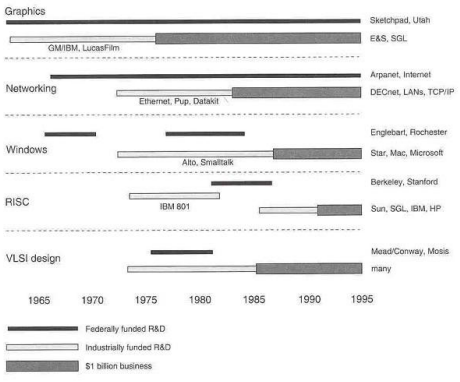


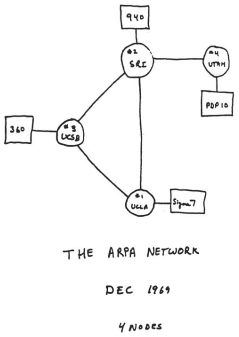
FIGURE II.13 Technological developments in computing.

Utah’s Historical Impact

Initial ARPANET:

1. Consisted of “Interface Message Processors” (IMP)
2. IMPs (routers) connected to local computer
3. Four nodes: Utah, UCLA, SRI, UCSB
4. Functional on 5 Dec. 1969

Utah’s IMP connected to PDP-10 running TENEX



THE ARPA NETWORK

DEC 1969

4 NODES

Utah's Historical Impact

Len Klienrick and the first IMP processor

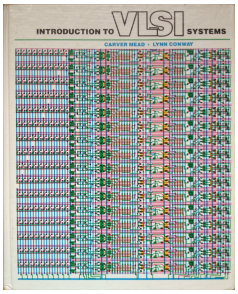
Async ILLIAC II modules

Utah spinoff:

CIRRUS LOGIC

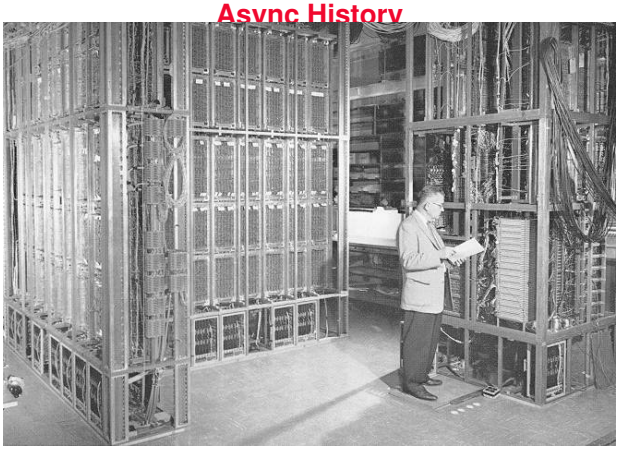
Utah's Historical Impact

Utah was at the center of Computer Engineering and microelectronics.



"Work on this text began in **August 1977**. The first three chapters were used as course notes during the fall of 1977, in courses given by Carver Mead at Caltech and by Carlo Sequin at U. C. Berkeley. [...] This printing is being used again during the **fall of 1978** in the courses at Caltech and U.C. Berkeley, and in new courses by Lynn Conway, while visiting at M.I.T., by **Kent Smith at University of Utah**, and Bob Bower at U.C.L.A."

Async History



ILLIAC II shown with no cover

History of Async Design

Just a few of the interesting players:

- Al Davis
  - Current Utah Professor
  - DDM1-3, FAIM
- Ken Stevens
  - Current Utah Professor
  - Burst-Mode, Post Office, Intel RAPPID, Relative Timing
- Erik Brunvand
  - Current Utah Professor
  - High Level Language for Async design
- Chris Myers
  - Current Utah Professor
  - Published "Asynchronous Circuit Design"

Design History

Circuits were initially designed using relays and vacuum tubes.

- large variability, unreliable components
- large σ gives horrible performance
- need to easily debug, repair

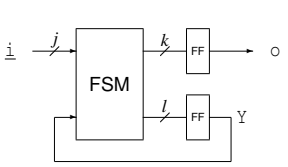
Hence circuits were largely asynchronous

Early work by founding fathers of circuit design

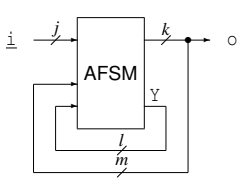
- Huffman: Fundamental mode circuits, Huffman machines
- Muller: speed-independent circuits

Design History

Synchronous (Mealy, Moore) FSM vs Asynchronous (Huffman) FSM:



(a) Clocked Finite State Machine



(b) Async. Finite State Machine

Design History

Totally disruptive technology: **Transistors!**

- Fairchild Semiconductor born (planar processing)
- Robust components
- Little variation
- Exceptional scaling potential
- ECL $\rightarrow$ TTL $\rightarrow$ NMOS $\rightarrow$ CMOS

Design History

Tightly controlled transistors allowed alternate design methodology: **clocking**

This gave significant advantages:

- Simple way to implement sequencing
- Simple approach to noise and *hazards*
  - The AFSM modules limited to FLOP and LATCH cells
  - Everything else combinational logic

Design History

But clocking also had a number of significant drawbacks:

- Need to distribute a clock
- Worst-case delays
- Guard banding to PVT variation
- Not modular
- Power issues
- Global pipeline dictates performance

Design History

Early revolutionary and high performance machines all asynchronous

- ILLIAC (1952) and ILLIAC2 (1962) – U. of Illinois
- Atlas (1962) and MU-5 (1966) – U. of Manchester
- Macromodules (60's – 70's) – Washington U., St. Louis
- Commercial Graphics (70's) – UofU, Evans & Sutherland
- DDM dataflow computer (78) – UofU

Design Methodology History

During this time frame:

- Effectively no CAD
  - circuits designed by hand on Mylar sheets
- System design was easy
  - formal, modular interfaces reactive to environment
- Module design was hard
  - limited to *single input change* SIC AFSMs
  - hand design of hazard-free circuits!!!

Hard aspect of design was easy
easy aspect of design was hard

Design History

Clocked design made easy part of design trivial

- No need to design sequentials
  - flops are all we need!
- No need to worry about hazards in combinational logic
- Synthesis now became a tractable problem
- Performance loss not as big an issue due to scaling and time to market advantages

Basically spelled the doom of asynchronous design

Design History

Some hardy fools continued to work in this area

- Most formal verification traces roots back to sequential verification of asynchronous systems

Significant designs built in *dead zone*
Domain of the independent and reckless:

- Post Office (1988) – HP (1.2Gb/s network)
- Fully asynchronous microprocessor (1989) – Caltech
- First code-compatible processor (1994) – U. Manchester

CAD History

CAD and methodology became a huge enabler for all design methodologies in the 1980's

- Async design became automated and more capable
  - Burst-Mode (1986), Stevens @ HP
    - Multiple input change AFSM
    - Extensions and tools developed at Stanford
  - ATACS (1992), Myers @ Stanford
    - formal design verification
    - included metric timing
  - Tangram (1990's), Eindhoven Netherlands
  - Petrify (1990's), Russia/Spain

Modern Era

Drawbacks of clocked design becoming increasingly frustrating

- Serious commercial research or design started
- Various concerns forcing issue
  - Clocks "hitting the wall"
  - Power dissipation
  - Radiation spectrum of clocked designs
  - Tough system design tradeoffs: modularity
  - SoC and multi-frequency designs

Modern History

Significant modern asynchronous chips

- Network Backplane (1990's) – Myracom
- Commercial pagers (1990's) – Philips
- Pentium front end (1998) – Intel
- Smart Cards (00's) – Philips
- Network Chips (05) – Fulcrum
- NoC Chips (05) – Silistix
- Desynchronization (07) – Nanochronous
- FPGA Chips (08) – Achronix
- Embedded FPGA (10) – Wave Semi

Asynchronous Millionaires?

Significant Commercial Startups today:

- Myracom (1990's) – MIT startup
  - Network backplanes
- Fulcrum Microelectronics (2005) – 30+ engineers – LA
  - Acquired by Intel 19 July 2011

Fundamental Properties of Async Circuits

1. FORMAL handshake PROTOCOLS
   a. Data validity
   b. Stall

   Normally use a request - acknowledge protocol
2. Reactive (not sampled)
   a. Hazards are not permitted
      - hazards indistinguishable from protocol transitions
3. Robust behavior, variable timing
   a. Can wait for metastability to resolve
   (crocked design: consistent timing but unreliable behavior)

Design Axes

VLSI has three inter-related design axes:

1. Functionality
   - Boolean logic, analog logic, . . .
2. Physical domain
   - Two dimensional wafers, process, resistance, cap, rise and fall times, etc.
3. Timing domain
   - Clocked, asynchronous, relative timed, . . .

Timing Domain - the Third Dimension

- **Delay Insensitive** (DI): Unbounded wire and gate delay
  - timing is *irrelevant* if physical circuit properties hold
  - no such circuits in reality. . .
- **Quasi Delay Insensitive** (QDI): DI with some "isochronic" wires
- **Speed-Independent** (SI): Unbounded gate delay, negligible wire delay
  - Wire forks are assumed to be *isochronic*: indistinguishable delay variation between devices on a wire
- **Fundamental Mode**: Circuit stabilizes between input transitions
  - Huffman SIC circuits and Burst-Mode circuits

Timing Domain - the Third Dimension

- **Equipotential Regions**
  - Delay variation between wires and logic is bounded
- **Self Timed**: timing kept inside circuit elements
  - Unspecific assumptions necessary for circuit to operate properly
  - Necessary for some protocols

Modular Asynchronous Protocols

- **Dual Rail**
  - Two wires encode a single data bit
  - A delay insensitive protocol
    - 00: spacer or NULL token
    - 01: valid 1
    - 10: valid 0
  - Data busses move from fully invalid to fully invalid
  - Simple to encode and decode (C-element)
  - Naturally half shielded
  - Data activity factor: 200%!!!

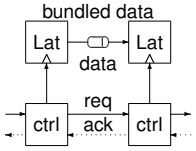
Modular Asynchronous Protocols

- n-of-m Codes
  - n signals asserted for m wires
  - A delay insensitive protocol
  - 1-of-4
    - 0000: spacer or NULL token
    - 0001: valid binary 0
    - 0010: valid binary 1
    - 0100: valid binary 2
    - 1000: valid binary 3
  - Same number of wires as Dual Rail
  - More difficult to encode and decode
  - Data activity factor: 100%

Modular Asynchronous Protocols

Bundled Data Protocols

- Standard data-path logic
  - combinational logic
  - can have hazards
  - max frequency analysis
- Careful design of sequential control
- Various handshake protocols
  - no min-delay races due to protocols
- Equipotential region for data/req race



Modular Asynchronous Protocols

Three major optimization points of bundled protocols

1. Control Protocol
   - 4-Cycle, 2-Cycle, . . .
   - Active or passive channels
2. Data
   - Broad, narrow, etc.
3. Timing
   - Fundamental mode, burst mode
   - Relative timing

Data Protocols

Communication Interface: called a **channel**

- Simple back-and-forth transitions on req and ack
- Data protocol part of channel

Bundled Data on the Channel:

1. Data direction
   - data can go either or both directions in a pipeline
   - don't need additional control signals
2. Push or pull / active or passive channel interface
   - determines if the data is *forwarded* or *requested*

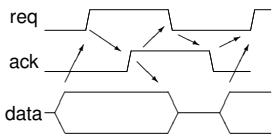
Data Protocols

3. when bundled data is valid:
   - When data becomes valid
     - rising or falling edge of req (4-phase protocol)
     - "early" or "late"
   - When data becomes invalid
     - "wide" or "narrow" protocol
     - protocol is wide if data is asserted *entire* handshake cycle while both request and acknowledge are asserted.
     - narrow protocol is when it is asserted from $\frac{1}{4}$ to $\frac{3}{4}$ of the full cycle
   - Determines how and when to latch data
     - does data need to be latched in broad protocol?

Modular Asynchronous Protocols

4-cycle return-to-zero protocol

- Most common protocol
- Two control wire transitions per data
- Logic and control state is *level sensitive*
  - channel active when any signal high

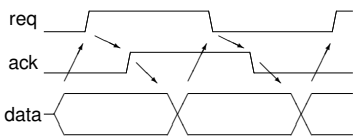


4-cycle protocol, early narrow data

Modular Asynchronous Protocols

2-cycle non-return-to-zero protocol

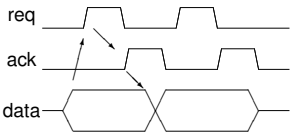
- Less common protocol
- One control wire transition per data
- Logic and control state is *edge sensitive*
  - channel active when req $\oplus$ ack
- Tradeoff of more complex logic (xors) with fewer control cycles
  - best for long channel communication latency



Modular Asynchronous Protocols

Pulse-mode protocol

- Pulses can be "captured" inside logic
- Contains properties of both 2- and 4-cycle circuits
  - level sensitive logic
  - no "redundant" handshakes
- Care must be taken to ensure pulses propagate
  - pulses are minimum size (3–5 FO4 delays)
  - wires are pulse filters
- Cannot easily determine state by observing channel wires



Modular Asynchronous Protocols

Source Synchronous Protocol

- Combination of *wave pipelining* and pulse-mode circuits
  - no *ack* handshake signal!
  - data is latched at receiver
  - pulse and spacing is usually bigger – "clock-like"
- Timing issues reduced by slower clock
- Overflow controlled at system level or at larger granularity

Asynchronous Control Circuits

Sequential Asynchronous Finite State Machines (AFSM)

- Specifications are *formal*
  - Finite state machine description – Burst-Mode
    - familiar format to engineers
  - Petri Net
  - Process Logic
- Circuits synthesized
  - MEAT & 3D (burst-mode), Petrify (DI/SI/Timed), ATACS (SI/Timed)

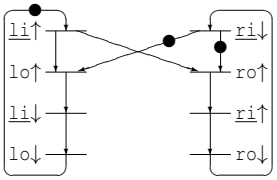
Asynchronous Control Circuits

AFSM Synthesis (cont)

- Mapped to circuits
  - domino is ultra efficient (power and performance)
  - static or complex gates
  - mapping can introduce hazards!!
- Verified for correctness
  - Spin, Analyze, . . .
  - relative timing - use Analyze-RT

Relative Timed Pipeline Controller

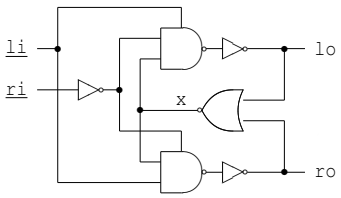
4-cycle linear RT handshake controller



$L = \underline{li}.\overline{\heartsuit}.lo.\underline{li}.lo.L$

$R = \quad \heartsuit.ro.\underline{ri}.ro.\underline{ri}.R$

$CTL = (L|R)\backslash\{\heartsuit\}$



Pipeline Controllers

Properties of interest in a pipelined controller

1. Circuit Performance
   - forward (data) latency
   - backward (bubble) latency
   - data throughput
2. Data Path Interface
   a. number of data items stored per latch
   b. amount of extra logic for latch control
3. Power Considerations
   a. normally open or closed latches
   b. leakage
4. Protocol and Timing
   a. is timing explicit in protocol?
   b. concurrency and protocol overhead

Mathematical Approach

Study mathematical properties of protocols:

1. Study pipelined configurations
   a. Pipelined in linear series
   b. Series and parallel linear pipelines
      - fork into series pipelines
      - join outputs back to single stream
2. Evaluate effects on interface protocols
   a. Adjacent interactions directly controlled by protocol
   b. Interfaces tend toward protocol fixed points

Formal Model of Pipeline Concurrency

Rules for protocol family investigated

- 4-phase protocol
- data valid before rising request

Rules:

s0: *Liveness:* quiescent state always reachable

s1: Data valid before rising input channel request

s2: Data captured before input channel ack rises

s3: Data through latch before output channel req rises

s4: Latch closed until output ack rises

Prove with latch specification that serially handshakes with controller

$$LATCH \;=\; \uparrow rEn.\,\mathrm{open}.\uparrow\overline{aEn}.\downarrow rEn.\,\mathrm{closed}.\downarrow\overline{aEn}.\,LATCH$$

Slow design, but lovely formal model

Formal Model of Pipeline Concurrency

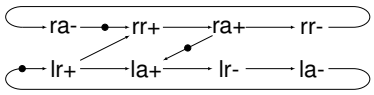
1. Liveness property **s0** holds due to concurrency reduction rules (coming soon)
2. Process L ensures **s2** holds (input data integrity)
3. Process V ensures **s3** holds (output data validity)
4. Process S ensures **s4** holds (output data integrity)
5. Protocol timing **s1** (input data validity) holds when **s3** true upstream

$$\begin{aligned}
L &= \uparrow lr.\,gS.\uparrow\overline{rEn}.\uparrow aEn.\downarrow\overline{rEn}.\downarrow aEn.\,pV.\uparrow\overline{la}.\downarrow lr.\downarrow\overline{la}.L\\
R &= gV.\uparrow\overline{rr}.\uparrow ra.\,pS.\downarrow\overline{rr}.\downarrow ra.R\\
S &= \overline{gS}.\overline{pS}.S \qquad\qquad V = \overline{pV}.\overline{gV}.V\\
LC &= (L \mid R \mid S \mid V)\setminus\{gS, pS, gV, pV\}\\
LATCH &= \uparrow rEn.\,\mathrm{open}.\uparrow\overline{aEn}.\downarrow rEn.\,\mathrm{closed}.\downarrow\overline{aEn}.\,LATCH\\
max &= (LC \mid LATCH)\setminus\{rEn, aEn\} \qquad (1)
\end{aligned}$$

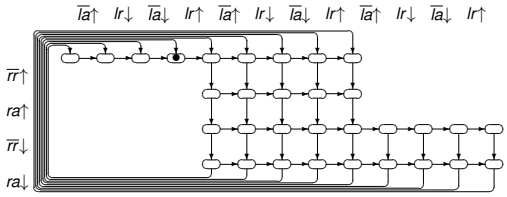
Executive summary: *The serialization of the latch handshake in process L allows us to prove correctness of abstraction and concurrency reduction approach.*

Maximum Concurrency Abstracted Protocol

As a petri net (very much like process on previous foil):

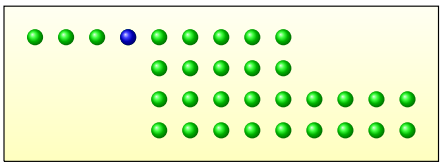


As minimized state graph, in particular configuration called **shape**



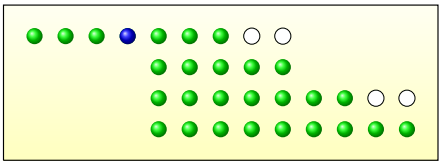
Concurrency Reduction (*Cuts*) from Most Concurrent Design (*Shape*)

- *Shape* is representation of state space in particular configuration
- Arcs not shown for clarity
- Reduce concurrency by removing states from left or right of rows



Concurrency Reduction (*Cuts*) from Most Concurrent Design (*Shape*)

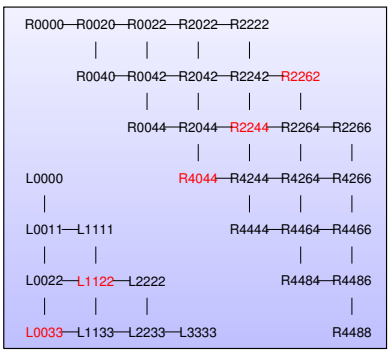
- *Shape* is representation of state space in particular configuration
- Arcs not shown for clarity
- Reduce concurrency by removing states from left or right of rows



Removed (or **cut**) 2 states from right of first and third rows

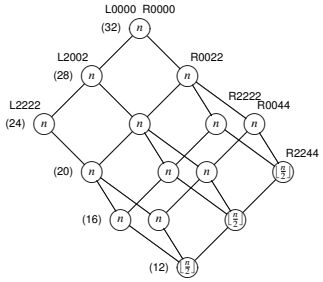
Correctness of Concurrency Reduction Rules

1. Complete and Coherent Symmetric Lattices
2. Each cut has complement around self-symmetric axis:
   - $L3333 = L(a_0+d_1)(b_0+c_1)(c_0+b_1)(d_0+a_1)$ $\quad R4488 = R(a_0+b_1)(b_0+a_1)(c_0+d_1)(d_0+c_1)$



Complete Untimed Protocol Family

- Certain properties fully defined by cuts
  - buffering capacity
  - behavior of homogeneous compositions
    - series pipelines
    - parallel pipelines



Complete Untimed Protocol Family

Observations:

1. Shown correctness and completion of abstraction and protocol family
2. Shown equivalence of multiple protocols in structured pipelines
3. All implementations in family can be abstracted to these protocols
4. Few of these protocols have been published
   - $\approx$ 40 of 137 untimed pipelined protocols

Questions:

What is the impact of concurrency reduction on designs?

Can we determine which protocol in the family is best?

What is the best implementation for equivalent protocols when pipelined?

Concurrency Reduction

Complex interplay between:

- Concurrency reduction simplifies logic
- Simpler logic assumed to be faster
- Concurrency reduction adds protocol delays
- Hazard removal and technology mapping creates “noise”

Our expectation:

- **optimal will lie somewhere in middle of concurrency reduction spectrum**

Will this hold globally as well as across protocol equivalent classes?

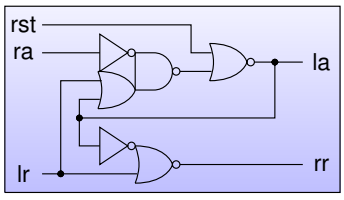
Circuit Complexity and Concurrency Reduction

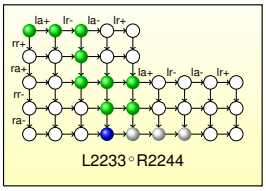
Number of explicit state variables generated by Petrify

L0000	L0011	L1111	L0022	L1122	L0033	L1133	L2222	L2233	L3333	L ∘ R
–	–	2	4	–	–	2	2	2	2	R0000
3	–	2	2	2	2	1	2	1	1	R0020
2	2	2	2	2	2	1	2	1	1	R0040
3	3	2	1	2	2	1	2	1	1	R0022
2	2	2	–	2	2	1	2	1	1	R0042
2	2	2	2	2	2	1	2	1	.	R2022
2	1	1	1	1	1	0	1	0	.	R2042
3	2	2	2	2	2	1	2	1	1	R0044
2	2	1	1	1	1	0	1	0	.	R2044
2	1	.	1	.	1	.	.	.	.	R4044
2	2	2	2	2	2	1	2	1	.	R2222
2	1	1	2	1	1	0	1	0	.	R2242
2	1	1	1	1	.	.	1	.	.	R2262
2	1	1	1	1	1	0	1	0	.	R2244
2	1	1	1	0	.	.	1	.	.	R2264
2	1	.	1	.	1	.	.	.	.	R4244
1	0	.	0	.	.	.	.	.	.	R4264

- “–”: no solution found “.”: deadlocking agent
- Complexity reduction assumption: generally holds

Area and Concurrency Reduction





Circuit with smallest area and backward latency

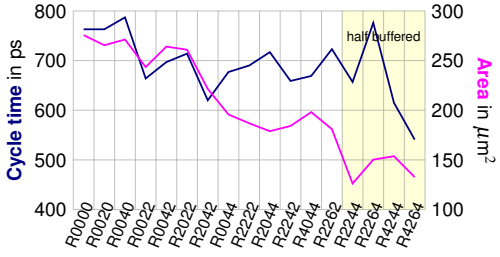
- half-buffered
- protocol L2233 ∘ R2244
- only three gates

Concurrency: Performance vs Parallelism

Assumption: $\exists$ a tradeoff between performance and parallelism

- Performance increases with concurrency reduction
- Protocol parallelism decreases with concurrency reduction

Evaluate comparing performance (area) with parallelism (cycle time)

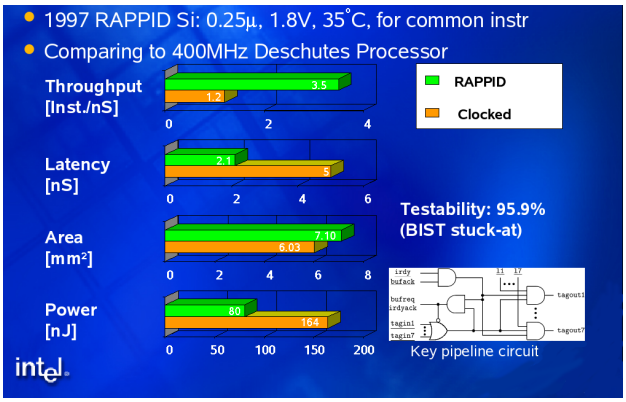


Asynchronous Advantages

The Pentium Processor Front End:

- Research Async vs Production Product
- Fabricated on same 180nm Intel process
- 4-cycle and pulse-mode channels
- Testability evaluated after design: 95.6% stuck-at testable

Timed Asynchronous Designs



Asynchronous Advantages

Improvement of async Pentium front end over commercial clocked design:

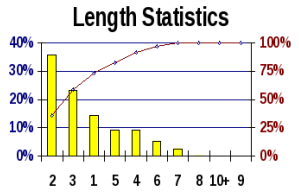
1. Throughput: 3× the clocked circuit
2. Energy per instruction: 2× improvement (half the energy)
3. Latency: 2× improvement (half the latency)
4. Area: 0.8× improvement (slightly larger layout)

Aggregate 10× improvement over clocked design

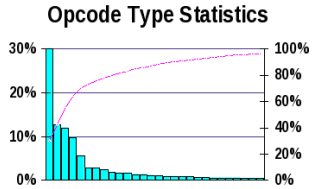
- Async frequency would be ∼ 40 GHz on today's 65nm process!

Optimize for Common Case

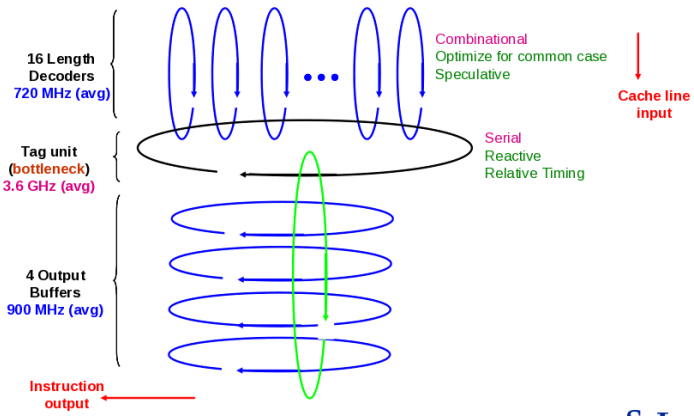
- Common Lengths:
  99.8% of the instructions are of length 7 or shorter



- Common Opcodes:
  8% of opcode types are used 80% of the time

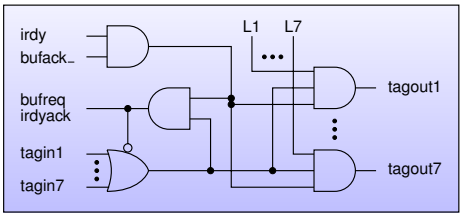


Multi-frequency Design is Key



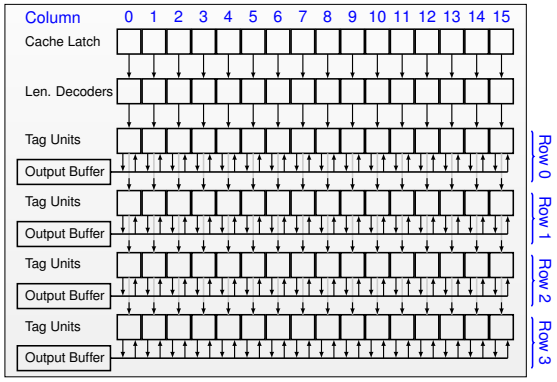
Concurrency and Time

- **System level timing** allows substantial simplification of circuits
- Multi-frequency property key aspect of timing optimizations
- Pentium Processor front end was key architectural experiment
  - Following circuit key concurrency and synchronization point
    - Note 7-input dynamic domino OR gate
  - Allowed 3GHz operation when clocked design at 500MHz (Deschutes)



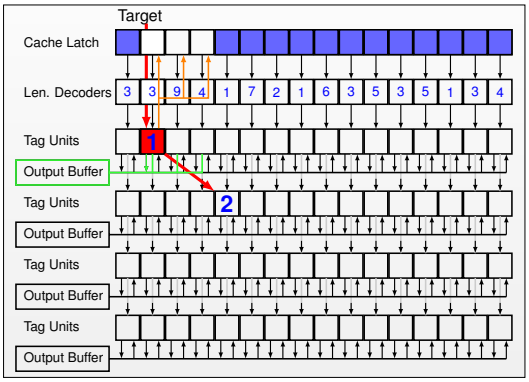
Concurrency and Time

Architectural level timing experiment: Pentium front end



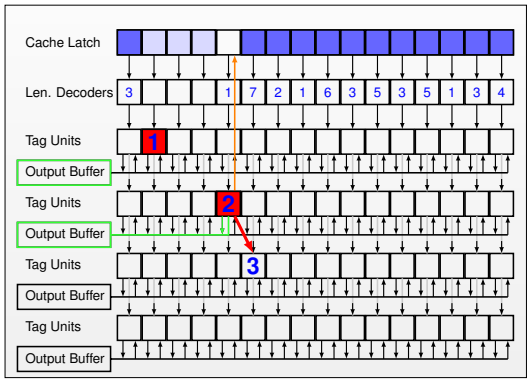
Concurrency and Time

Architectural level timing experiment: Pentium front end



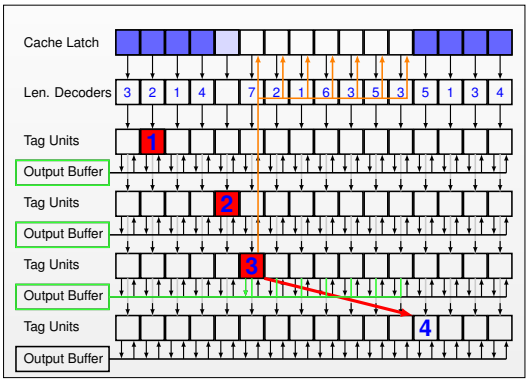
Concurrency and Time

Architectural level timing experiment: Pentium front end



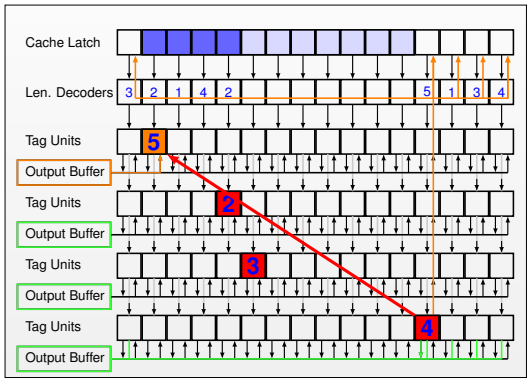
Concurrency and Time

Architectural level timing experiment: Pentium front end



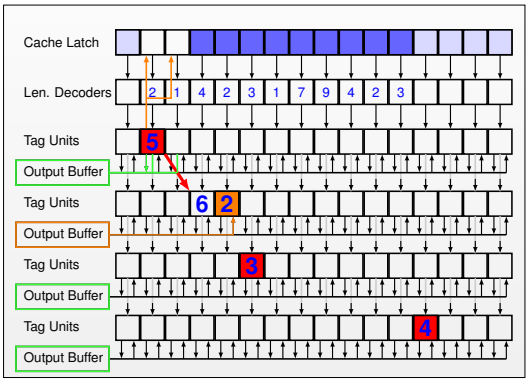
Concurrency and Time

Architectural level timing experiment: Pentium front end



Concurrency and Time

Architectural level timing experiment: Pentium front end



Async Review

1. what is an asynchronous circuit
2. what are the main asynchronous circuits in use in all our chips today?
3. how did transistors effect design methodologies
4. advantages of asynchronous design
5. disadvantages of async design
6. advantages of clocking
7. disadvantages of clocked methodologies
8. where would you design with clocked or asynchronous designs?
9. three fundamental properties of Async circuits

Async Review

1. how has CAD affected async design
2. common concerns currently motivating async design
3. timing assumptions of DI, QDI, SI, fundamental mode
4. which protocols are delay insensitive?
5. channel concept
6. bundled data
7. 4-cycle, 2-cycle, pulse-mode and source synchronous protocols
8. general data alignment in async protocols
9. which gate family is low power in asynchronous circuits? why?