Outline

- Terminology
- Motivation
- Asynchronous Micro-pipelines
  - The bulk of this lecture
- Bundled Data Designs
- Pro’s and Con’s of Asynchronous Design for Processors
  - Facts and fallacies
- Other Uses of Asynchronous Design
- Arbiters and Synchronizers
- Further Reading
Terminology

• Asynchronous design deals with signals that are not guaranteed to transition within an interval specified relative to a (system) clock
  - Ranges from signals with unknown clock phase (common in latency-tolerant I/O) to signals that are synchronous to other clocks, to signals that have no periodicity whatsoever (coming from you?)
• Self-timed circuits are circuits that operate (at least locally) without a (system) clock
  - Examples: self-resetting logic (Lecture 11), (reset locally derived instead of derived from clock), one shot (pulse gen.)
• Delay-insensitive (DI) circuits are circuits that operate correctly no matter what the delay of the component gates and/ or wires that interconnect them
• Timed circuits are circuits whose correct operation depends on timed behavior (and thus analysis)
Motivation

• Reasons to avoid clocks altogether:
  - Clocks don’t do useful computation
  - Clock uncertainty penalizes cycle time
  - Clocks and latches take a lot of power
    • More than 50% in high-frequency designs
  - What are latches for anyway?

• Let us go all-out and first look at some circuits that do not require any clocks (but do sequencing none the less)
Basic Component of Self-Timed Circuits: Muller C-Element

<table>
<thead>
<tr>
<th>i0 \ i1</th>
<th>F</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>y</td>
</tr>
<tr>
<td>T</td>
<td>y</td>
<td>T</td>
</tr>
</tbody>
</table>

The Truth Table

A CMOS Implementation

C-element Symbol
Self-Timed Micropipeline
(first without data, just events)

Rules:
• Transitions on L_i and L_o strictly alternate
• Transitions on R_o and R_i strictly alternate
• Input event is L_i,L_o transition pair (2-phase)
• Output event is R_o,R_i transition pair (2-phase)
• One output event per input event

Slack:
Max. number of input events that can be completed before an output event is completed
0-Slack Pipeline

Active input: \( L_i; R_o; R_i; L_o \)
or
Active output: \( R_i; L_o; L_i; R_o \)

(Each \( L_x \) or \( R_x \) indicates a transition (0->1 or 1->0))
Slack 1 (2-phase) Self-Timed Pipeline

(Ivan Sutherland)

Initial State (State 0)
Slack 1 (2-phase) Self-Timed Pipeline
(L active, R passive)

\[ L_i; \]

Output is “enabled”
Slack 1 (2-phase) Self-Timed Pipeline

L_i; L_o; R_o;
One input event (L_i; L_o) completed

Next: BOTH R_i and L_i may transition
ORDER DOES NOT MATTER
(obey rules either way)
Both must transition before output transitions
Slack 1 (2-phase) Self-Timed Pipeline

L_i; L_o; R_o; R_i; L_i;

One output event (R_o; R_i) completed

Output is “enabled”
Slack 1 (2-phase) Self-Timed Pipeline

\[ L_i; L_o; R_o; R_i; L_i; L_o; R_o; \]

Two Input events completed \( (L_i; L_o; L_i; L_o) \)
One Output event completed \( (2^{nd} \text{ enabled}) \)

Both \( R_i \) and \( L_i \) can transition
If \( R_i \) transitions back in State 0
Slack 2 (2-phase) Self-Timed Pipeline

(L Active, R Passive)
Slack 2 (2-phase) Self-Timed Pipeline

(L passive, R Active)

Alternatively, can simply change initial state to go from active in to active out
Slack 1 (4-phase) Self-Timed Pipeline

(L Active, R Passive)

4-phase: L_i(up); L_o(up); L_i(down); L_o(down)

Why ever do 4-phase (half as efficient)?
Answer: Computation in CMOS is easier with level_sensitive circuits.
One-Bit Slack \( \frac{1}{2} \) (4-phase) Self-Timed Buffer (Quasi Delay-Insensitive (QDI))

\[ \text{L}_0 \text{i} \quad \text{L}_1 \text{i} \quad \text{L}_0 \text{o} \quad \text{R}_0 \text{o} \quad \text{R}_1 \text{o} \quad \text{R}_i \]
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

L0_i \quad R0_o

L_o \quad R_i

L1_i \quad R1_o

L1_i(up);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

L0_i \quad R0_o
L_o \quad R_i
L1_i \quad R1_o

L1_i(up); L_o(up);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

$L0_i \quad L1_i \quad L_o \quad R0_o \quad R1_o \quad R_i$

$L1_i(\text{up}); L_o(\text{up}); L1_i(\text{down});$
One-Bit Slack ½ (4-phase) Self-Timed Buffer

L0_i \quad R0_o

L_o \quad R_i

L1_i \quad R1_o

L1_i(up); L_o(up); L1_i(down); R1_o(up);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

L1_i(up); L_o(up); L1_i(down); R1_o(up);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

L0_i

L_o

L1_i

R0_o

R_i

R1_o

L1_i(up);L_o(up);L1_i(down);R1_o(up);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

$L1_i \text{(up)} ; L_o \text{(up)} ; L1_i \text{(down)} ; R1_o \text{(up)} ; L_o \text{(down)} ;$
One-Bit Slack 1/2 (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

L_{0_i}, L_{o}, L_{1_i}, R_{0_o}, R_{i}, R_{1_o}

L_{1_i}(up); L_{o}(up); L_{1_i}(down); R_{1_o}(up); L_{o}(down); R_{i}(up);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

L1_i(up);L_o(up);L1_i(down);R1_o(up);L_o(down);
R_i(up);R1_o(down);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

L0_i, R0_o

L_i, R_i

L1_i, R1_o

L1_i(up); L_o(up); L1_i(down); R1_o(up); L_o(down);
R_i(up); R1_o(down);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition
(potential hazard)

L1_i(up); L_o(up); L1_i(down); R1_o(up); L_o(down);
R_i(up); R1_o(down); R_i(down);
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Once more, now doing all transitions as soon as enabled (and counting FO4, 2 per gate)
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

$L_0_i, L_0_o, L_{1_i}, R_{0_o}, R_{1_o}$

$L_{1_i}$(up); 2 FO4 (assume env. is another stage)
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

L0_i

L_o

L1_i

R0_o

R_i

R1_o

L1_i(up);
4FO4
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

L0_i

L_o

L1_i

R0_o

R_i

R1_o

L1_i(up); L_o(up); R1_o(up)

6FO4
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

$\overline{L_0}_i \overline{L_o} \overline{L1}_i \overline{R0}_o \overline{R_i} \overline{R1}_o$

$\overline{L1}_i(up); \overline{L_o}(up); \overline{R1}_o(up); \overline{L1}_i(down); \overline{R_i}(up);
8FO4
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition (potential hazard)

L0_i \quad R0_o
L_o \quad R_i
L1_i \quad R1_o

L1_i(up); L_o(up); R1_o(up); L1_i(down); R_i(up); R1_o(up); 10FO4
One-Bit Slack $\frac{1}{2}$ (4-phase) Self-Timed Buffer

Unacknowledged transition
(potential hazard)

L0_i \quad R0_o

L_o \quad R_i

L1_i \quad R1_o

L1_i(up); L_o(up); R1_o(up); L1_i(down); R_i(up);
R1_o(up); L_o(down); R1_o(down); 12FO4
(last transition go in parallel with next L0/1_i(up) )
A Few Observations

• No Clocks in Sight!
  – Yes, you can build a processor without clocks at all
• Many Interleavings Possible
  – Corresponding state explosion makes traditional state space exploration based verification tricky
• Unacknowledged transitions occur
  – None in event-pipe stage, one in 1-bit pipe stage
  – Can cause faulty operation (deadlock or spurious firing)
  – Cannot be avoided! Fully delay-insensitive are not Turing complete (Manohar, Martin)
  – Typically addressed by assuming (short) wires are faster than transistors (equipotential regions)
• FAST!
  – Cycle time is about 12FO4
  – Fastest self-timed pipelines are even faster (up to 6 FO4)
    • GASP (Sun, Async’2001), IPCMOS (IBM ISSCC’2000) and others
How About Computation?
Alternate Style (non-Delay Insensitive)

Bundled Data

- CLn must be faster than control
  - "Timed" circuit (correctness depends on timing analysis)
  - Can delay control to achieve this

- Used in Amulet microprocessors
  - Asynchronous ARM
  - … and many others
Pro’s and Con’s of Asynchronous Design for Processors

• Fallacy:
  - Clocking Overhead is Constant, we MUST use ASYNC soon

• Fact:
  - Clock generation and distribution techniques can still improve dramatically
    • High Q LC Oscillators reduce jitter, and are now feasible
    • Best clock distributions achieve design skew under 10ps
  - Most chips are still built with global clocking strategies
    • Penalizing “local” data transfers differently from “global” transfers promises another dramatic improvement (multiple clock domains predicted by ITRS roadmap)
  - New circuit approaches to clocking
    • Lecture 13, and this year’s ASYNC conference (SIC)

• BUT:
  - 6FO4 (demonstrated multiple times) is mighty fast
Pro’s and Con’s of Asynchronous Design for Processors

• Fallacy:
  - Asynchronous designs are always lower power

• Fact:
  - It is always performance/power (MIPS/W, Mips\(^2\)/W, Mips\(^3\)/W) that counts. Most comparisons I’ve seen ignore this fact. On the best experiment (Amulet vs. ARM core) Asynchronous designs have improved steadily vs. clocked designs, but are still a step behind.

• BUT:
  - Not bad for a small team! (Perhaps interesting to measure the class project against them …)
Pro’s and Con’s of Asynchronous Design for Processors

• Increased area for asynchronous design
  – But compared to fast dual-rail logic the difference is limited

• Average-case performance gives ASYNC an edge
  – But the slowest component (e.g. memory) can still set the “cycle” (if frequently used)

• ASYNC is more robust (device size variations)
  – Yes! (And device variability is increasing)

• ASYNC is more robust (SER events)
  – As long as the SER event merely delays a transition or stays below the threshold of one, ASYNC will not malfunction
  – Spurious transitions often lead to deadlock

• EMI spectrum of ASYNC processors is better
  – Usually the “cycle(s)” frequencies are still quite visible (see “average case”)
Other Uses of Asynchronous Design

• I/Os
  - Known frequency, unknown phase (can be clock with data)
    • Requires frequency locking mechanism (Lecture 6) or shared global clock, or a very stable reference
    • For high-speed I/O, one or more PLL/DLLs is typically required
  - Unknown frequency, or multiple unrelated clocks
    • Common in SOC designs
    • Potential for “metastability failure” (coming up ...)

• GALS
  - “Globally Asynchronous, Locally Synchronous”
  - Asynchronously Interfaced, Clocked Components
    • Components must be large enough to amortize synchronization overhead

• Locally Asynchronous Globally Synchronous
  - Self-resetting Dynamic Logic
    • Tricky (labeled “Do not use” in Lecture 11)
    • Fairly common in array designs
  - Asynchronous pipelines embedded in synchronous designs
    • Very effective if pipeline is “fast enough”
Arbiters and Synchronizers

- Not enough time to do the topic justice, but I want to mention a few things

- **WHENEVER A DEVICE (CIRCUIT) DETERMINES AN ORDER BETWEEN TWO SIGNALS THAT CAN ARRIVE IN EITHER ORDER OR AT THE SAME TIME, DETERMINING THE RESULT (ORDER) CAN TAKE AN ARBITRARILY LONG TIME TO COMPUTE**
  - Example: ASYNC signal against clock (Latching an ASYNC signal)
  - Example: Arbitration between two asynchronous signals
  - In a clocked system the possibility of metastability failure on resolving asynchronous signals is unavoidable
    - But the probability can be made arbitrarily small (decays exponentially with time)
    - Typically a few stages of specially designed latches reduces the probability of failure to acceptable levels
  - Can exchange uncertainty in signal level for uncertainty in time
    - Async arbiter circuit has indeterminate delay, but no spurious output transitions
Merge Element
(Quasi Delay-Insensitive (QDI))

Assumes A/B mutual exclusion guaranteed by environment
Mutual Exclusion Element
(cross coupled NAND followed by filter ckt.)

Output won’t transition until metastability is resolved.
4-phase Arbiter Circuit
(WRONG)
Simple Combination of Mutex and Merge

What is wrong with this circuit?
Arbiter Circuit
Combination of Mutex and Merge

La_o

La_i

Lb_i

Lb_o

R_i

R_o
Further Reading

- Proceedings of the International Symposium on Asynchronous Circuits and Systems (ASYNC) (eighth is going on now)
- Quite a few books have been written, but unfortunately there are many subtly different design methodologies, and books tend to be written from a rather specific perspective

A Final Prediction

- When the exponential VLSI technology improvement curve flattens out, a golden era of VLSI design will be next

You’ve been studying the right stuff! Good luck!