Previously

• Reviewed Pipelining
  – basic assignments on
• Saw spatial designs efficient
  – when reuse logic at maximum frequency
• Interconnect dominant delay
  – and dominant area
  – heavy call to reuse to use efficiently
Today

• Systematic transformation for retiming
  – maximize throughput
  – preserve semantics
  – “justify” mandatory registers in design

Motivation

• FPGAs (spatial computing)
  – run efficiently when all resources reused rapidly
    • cycle time minimized

• “Everything in the right place at the right time.”
Task

- Move registers to:
  - preserve semantics
  - Minimize path length between registers
  - (make path length 1 for maximum throughput or reuse)
  - Maximize reuse rate
  - …while minimizing number of registers required

Simple Example

Path Length ($L = 4$

Can we do better?
Legal Register Moves

• Retiming Lag/Lead

Canonical Graph Representation

Separate arch for each path
Weight edges by number of registers
(weight nodes by delay through node)
Critical Path Length

**Critical Path**: Length of longest path of zero weight nodes

Compute in $O(|E|)$ time by levelizing network:
- Topological sort, push path lengths forward until find register.

Retiming Lag/Lead

**Retiming**: Assign a lag to every vertex

$\text{weight}(e') = \text{weight}(e) + \text{lag}(\text{head}(e)) - \text{lag}(\text{tail}(e))$
Valid Retiming

- Retiming is valid as long as:
  - \( \forall e \) in graph
    - \( \text{weight}(e') = \text{weight}(e) + \text{lag(head}(e)) - \text{lag(tail}(e)) \geq 0 \)

- Assuming original circuit was a valid synchronous circuit, this guarantees:
  - non-negative register weights on all edges
    - no travel backward in time :-)
  - all cycles have strictly positive register counts
  - propagation delay on each vertex is non-negative (assumed 1 for today)

Retiming Task

- Move registers \( \equiv \) assign lags to nodes
  - lags define all locally legal moves
- Preserving non-negative edge weights
  - (previous slide)
  - guarantees collection of lags remains consistent globally
Retiming Transformation

• N.B. -- unchanged by retiming
  – number of registers around a cycle
  – delay along a cycle

• Cycle of length $P$ must have
  – at least $P/c$ registers on it
  – to be retimeable to cycle $c$

Optimal Retiming

• There is a retiming of
  – graph $G$
  – w/ clock cycle $c$
  – \textit{iff} $G-1/c$ has no cycles with negative edge weights

• $G-\alpha \equiv$ subtract $\alpha$ from each edge weight
Compute Retiming

• Lag(v) = shortest path to I/O in \( G - 1/c \)

• Compute shortest paths in \( O(|V||E|) \)
  – Bellman-Ford
  – also use to detect negative weight cycles when \( c \) too small
Bellman Ford

• For $I \leftarrow 0$ to $N$
  – $u_i \leftarrow \infty$ (except $u_i=0$ for IO)
• For $k \leftarrow 0$ to $N$
  – for $e_{i,j} \in E$
    • $u_i \leftarrow \min(u_i, u_j + w(e_{i,j}))$
• for $e_{i,j} \in E$
  • if $u_i > u_j + w(e_{i,j})$
    – cycles detected

Apply to Example
Apply: Find Lags

Apply: Lags
Apply: Move Registers

weight(e) = weight(e) + lag(head(e)) - lag(tail(e))

Apply: Retimed
Apply: Retimed Design

Revise Example (fanout delay)
Revised: C=1?

[Diagram with arrows and numbers]

Revised: C=2?

[Diagram with arrows and fractions]
Revised: Lag

Take ceiling to convert to integer lags:
Pipelining

- Can use this retiming to pipeline
- Assume have enough (infinite supply) of registers at edge of circuit
- Retime them into circuit
C>1 ==> Pipeline

Add Registers
Pipeline Retiming: Lag

\[ n=5 \]

\[ \begin{array}{cccccc}
4 & 0 & 0 & 0 & 0 & 0 \\
4 & -1 & -1 & -1 & -1 & -1 \\
0 & -1 & -1 & -2 & -3 & 0 \\
\end{array} \]

Pipelined Retimed

\[ n=5 \]

\[ \begin{array}{cccccc}
5 & 1 & 1 & 1 & 1 & 1 \\
4 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{array} \]
Real Cycle
Cycle C=1?

Cycle C=2?
Cycle: C-slow

Cycle=c ⇒ C-slow network has Cycle=1

2-slow Cycle ⇒ C=1
2-Slow Lags

![Diagram of 2-Slow Lags]

2-Slow Retime

![Diagram of 2-Slow Retime]
Retimed 2-Slow Cycle

C-Slow applicable?

- Available parallelism
  - solve C identical, independent problems
    - e.g. process packets (blocks) separately
    - e.g. independent regions in images

- Commutative operators
  - e.g. max example
Max Example

2-Slow design:

\[ X \longrightarrow \text{max} \longrightarrow Y \]

\[ X_2 X_1 X_1 X_0 X_0 \longrightarrow Y_2 \ ? \ Y_1 \ ? \ Y_0 \ ? \]

\[ B_2 A_2 B_1 A_1 B_0 A_0 \longrightarrow Y_{A_2} Y_{B_1} Y_{A_1} Y_{B_0} Y_{A_0} \ ? \]

Max Example

Computes two interleaved streams: even max, odd max

Computes final max of even and odd pairs
HSRA Retiming

- HSRA
  - adds mandatory pipelining to interconnect
- One additional twist
  - long, pipelined interconnect
  - \( \Rightarrow \) need more than one register on paths

Monday Lecture
Stopped Here
Accommodating HSRA Interconnect Delays

- Add buffers to LUT→LUT path to match interconnect register requirements
- Retime to \(C=1\) as before
- Buffer chains force enough registers to cover interconnect delays
Big Ideas
[MSB Ideas]

- Retiming important to
  - minimize cycles
  - efficiently utilize spatial architectures
- Optimally solvable in $O(|V||E|)$ time
- Tells us
  - pipelining required
  - C-slow
  - where to move registers