

## Overview

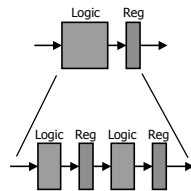
- ◆ Last lecture
  - State encoding
    - ☛ One-hot encoding
    - ☛ Output encoding
- ◆ Today:
  - Optimizing FSMs
    - ☛ Pipelining
    - ☛ Retiming
    - ☛ Partitioning
  - Conclusion of sequential logic

## Definitions

- ◆ Latency: Time to perform a computation
  - Data input to data output
- ◆ Throughput: Input or output data rate
  - Typically the clock rate
- ◆ Combinational delays drive performance
  - Define  $d \equiv$  delay through slowest combinational stage
  - $n \equiv$  number of stages from input to output
  - Latency  $\propto n \times d$  (in sec)
  - Throughput  $\propto 1/d$  (in Hz)

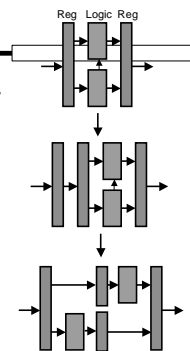
## Pipelining

- ◆ What?
  - Subdivide combinational logic
  - Add registers between logic
- ◆ Why?
  - Trade latency for throughput
    - ☛ Reduce logic delays
    - ☛ Increase clock speed
  - Increased latency
    - ☛ Takes cycles to fill the pipe
  - Increase circuit utilization
    - ☛ Simultaneous computations

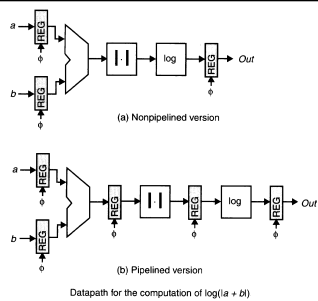


## Pipelining

- ◆ When?
  - Need throughput more than latency
    - ☛ Signal processing
  - Logic delays > setup/hold times
  - Acyclic logic
- ◆ Where?
  - At natural breaks in the combinational logic
  - Adding registers makes sense

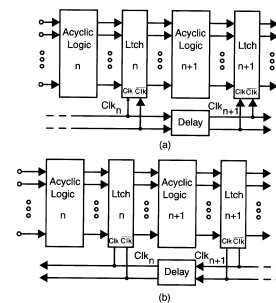


## Pipelining example



## Pipelining and clock skew

- ◆ Which is faster?
- ◆ Which is safer?

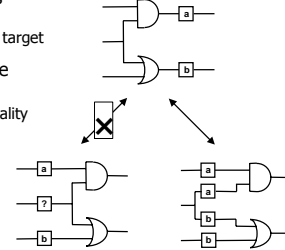


## Retiming

- ◆ Pipelining adds registers
  - To increase the clock speed
- ◆ Retiming moves registers around
  - Reschedules computations to optimize performance
    - ✔ Minimize critical path
    - ✔ Optimize logic across register boundaries
    - ✔ Reduce register count
  - Without altering functionality

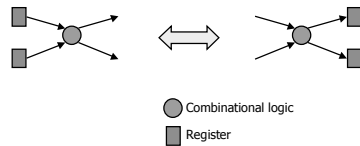
## Retiming in a nutshell

- ◆ Change position of FFs
  - For speed
  - To suit implementation target
- ◆ Retiming modifies state assignment
  - Preserves FSM functionality



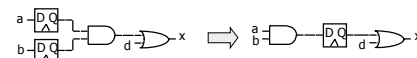
## Retiming groundrules

- ◆ Rules:
  - Remove one register from each input and add one to each output
  - Remove one register from each output and add one to each input

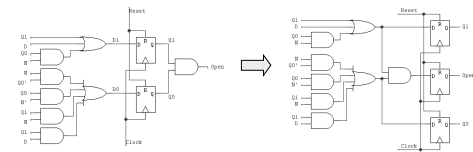


## Retiming examples

- ◆ Reduce register count

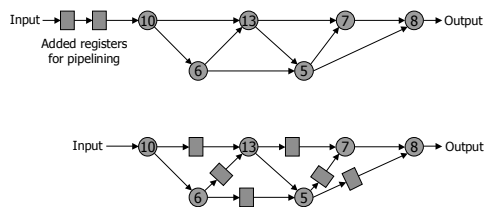


- ◆ Change output delays



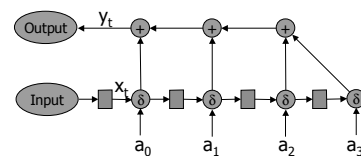
## Optimal pipelining

- Add registers
- Use retiming to optimize location



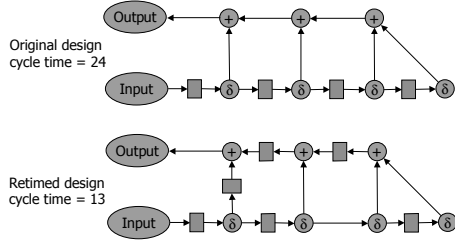
## Example: Digital correlator

- ◆  $y_t = \delta(x_t, a_0) + \delta(x_{t-1}, a_1) + \delta(x_{t-2}, a_2) + \delta(x_{t-3}, a_3)$ 
  - $\delta(x, a) = 1$  if  $x = a$ ; 0 otherwise



### Example: Digital correlator (cont'd)

- ◆ Delays: Comparator = 3; adder = 7



### FSM partitioning

- ◆ Break a large FSM into two or more smaller FSMs

#### ◆ Rationale

- Less states in each partition
  - ◆ Simpler minimization and state assignment
  - ◆ Smaller combinational logic
  - ◆ Shorter critical path
- But more logic overall

#### ◆ Goal

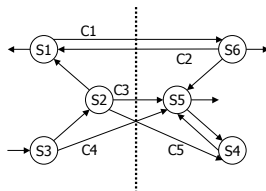
- Minimize communication between partitions
  - ◆ Minimize wires & I/O

#### ◆ Partitions are synchronous

- Same clock!!!

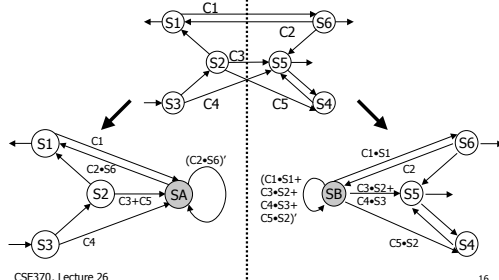
### Example: Partition the machine

- ◆ Partition into two halves



### Introduce idle states

- ◆ SA and SB handoff control between machines



### Partitioning rules

- Rule #1: Source-state transformation  
Replace by transition to idle state (SA)

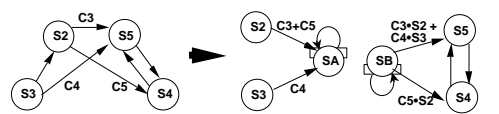


- Rule #2: Destination state transformation  
Replace with exit transition from idle state

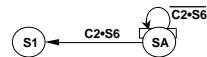


### Partitioning rules (cont')

- Rule #3: Multiple transitions with same source or destination  
Source  $\Rightarrow$  Replace by transitions to idle state (SA)  
Destination  $\Rightarrow$  Replace with exit transitions from idle state



- Rule #4: Hold condition for idle state  
OR exit conditions and invert

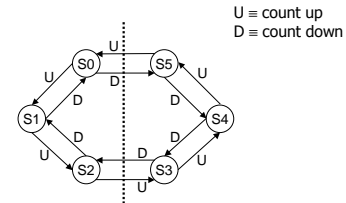


## Mealy versus Moore partitions

- ◆ Mealy machines undesirable
  - Inputs can affect outputs immediately
    - ⚡ "output" can be a handoff to another machine!!!
  - Inputs can ripple through several machines in one clock cycle
- ◆ Moore or synchronized Mealy desirable
  - Input-to-output path always broken by a flip-flop
  - But... may take several clocks for input to propagate to output
    - ⚡ Output may derive from other side of a partition

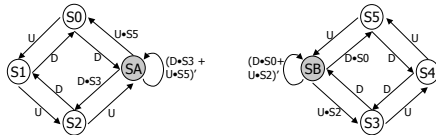
## Example: Six-state up/down counter

- ◆ Break into 2 parts



## Example: 6 state up/down counter (con't)

- ◆ Count sequence  $S_0, S_1, S_2, S_3, S_4, S_5$ 
  - $S_2$  goes to  $S_3$  and holds, leaves after  $S_5$
  - $S_5$  goes to  $S_4$  and holds, leaves after  $S_2$
  - Down sequence is similar



## Minimize communication between partitions

- ◆ Ideal world: Two machines handoff control
  - Separate I/O, states, etc.
- ◆ Real world: Minimize handoffs and common I/O
  - Minimize number of state bits that cross boundary
  - Merge common outputs
- ◆ Look for:
  - Disjoint inputs used in different regions of state diagram
  - Outputs active in only one region of state diagram
  - Isomorphic portions of state diagram
    - ⚡ Add states, if necessary, to make them so
  - Regions of diagram with a single entry and single exit point

## Sequential logic: What you should know

- ◆ Sequential logic building blocks
  - Latches (R-S and D)
  - Flip-flops (master/slave D, edge-triggered D & T)
  - Latch and flip-flop timing (setup/hold time, prop delay)
  - Timing diagrams
  - Flip-flop clocking
  - Asynchronous inputs and metastability
  - Registers

## Sequential logic: What you should know

- ◆ Counters
  - Timing diagrams
  - Shift registers
  - Ripple counters
  - State diagrams and state-transition tables
  - Counter design procedure
    1. Draw a state diagram
    2. Draw a state-transition table
    3. Encode the next-state functions
    4. Implement the design
  - Self-starting counters

## Sequential logic: What you should know

---

- ◆ Finite state machines
  - Timing diagrams (synchronous FSMs)
  - Moore versus Mealy versus registered Mealy
  - FSM design procedure
    1. Understand the problem (state diagram & state-transition table)
    2. Determine the machine's states (minimize the state diagram)
    3. Encode the machine's states (state assignment)
    4. Design the next-state logic (minimize the combinational logic)
    5. Implement the FSM
  - FSM design guidelines
    - ✦ Separate datapath and control
  - One-hot encoding
  - Pipelining and retiming basics