EE382
Processor Design

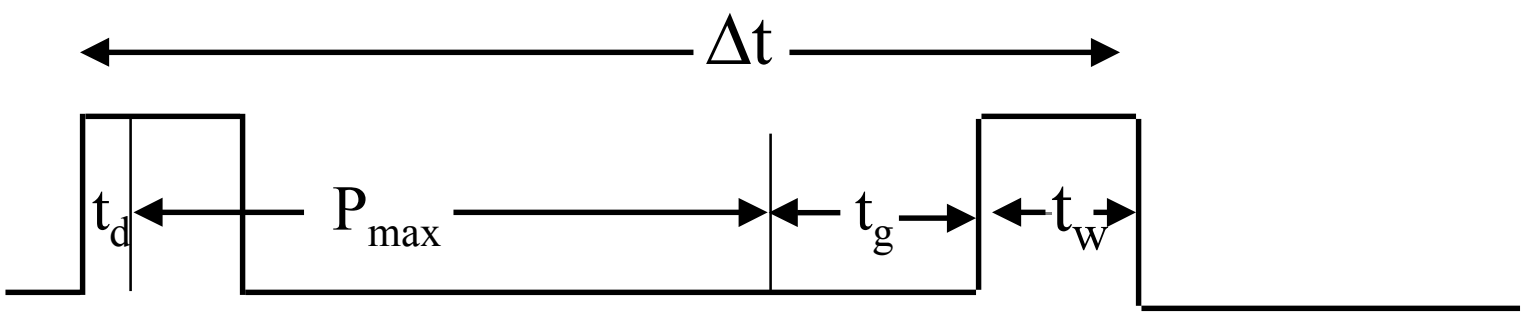
Winter 1999
Chapter 2 Lectures
Clocking and Pipelining

Topics

- **Clocking**
  - Clock Parameters
  - Latch Types
  - Requirements for reliable clocking
- **Pipelining**
  - Optimal pipelining
  - Pipeline partitioning
  - Asynchronous and self timed logic
  - Wave Pipelining and low overhead clocking

Clock Parameters

- **Parameters**
  - P_{max}- maximum delay thru logic
  - P_{min}- minimum delay thru logic
  - Δt - cycle time
  - t_w - clock pulse width
  - t_g - data setup time
  - t_d - register output delay
  - C - total clocking overhead



$$\Delta t = P_{max} + C$$

Latch Types

- **Cycle time depends on clock parameters and underlying latch**
  - edge- vs level-triggered
  - single- vs dual-rank

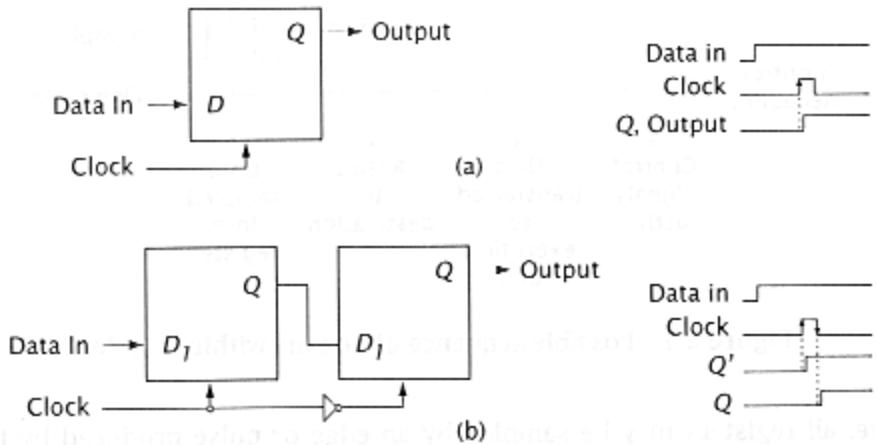


Figure 2.2 Edge-triggered clocks. (a) Single rank. (b) Dual rank.

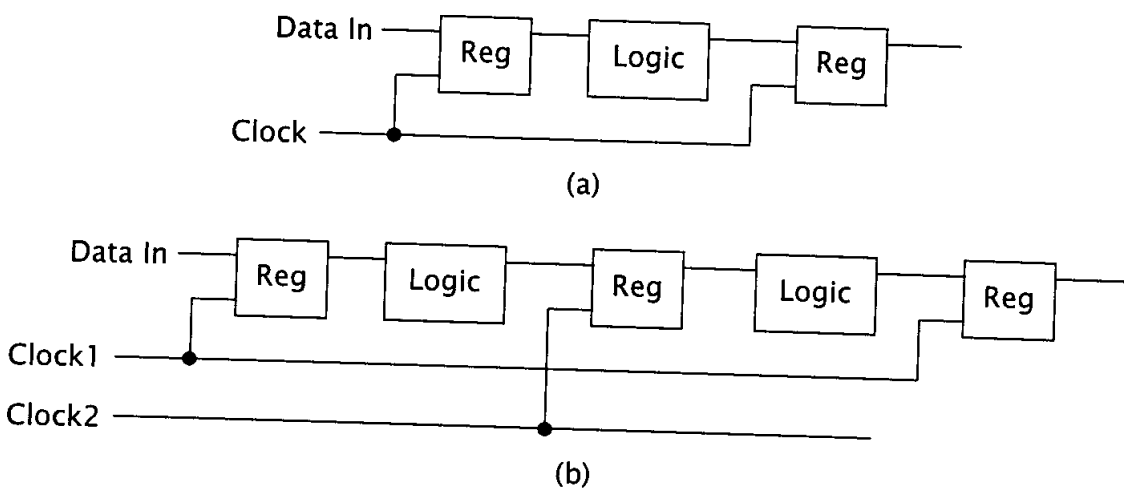


Figure 2.5 Clocks. (a) Single phase. (b) Two-phase.

Clock Overhead

Trigger/Rank	Single	Dual
Level	t_g+t_d	$2(t_g+t_d)$
Edge	t_g+t_d	$t_g+t_d+t_w$

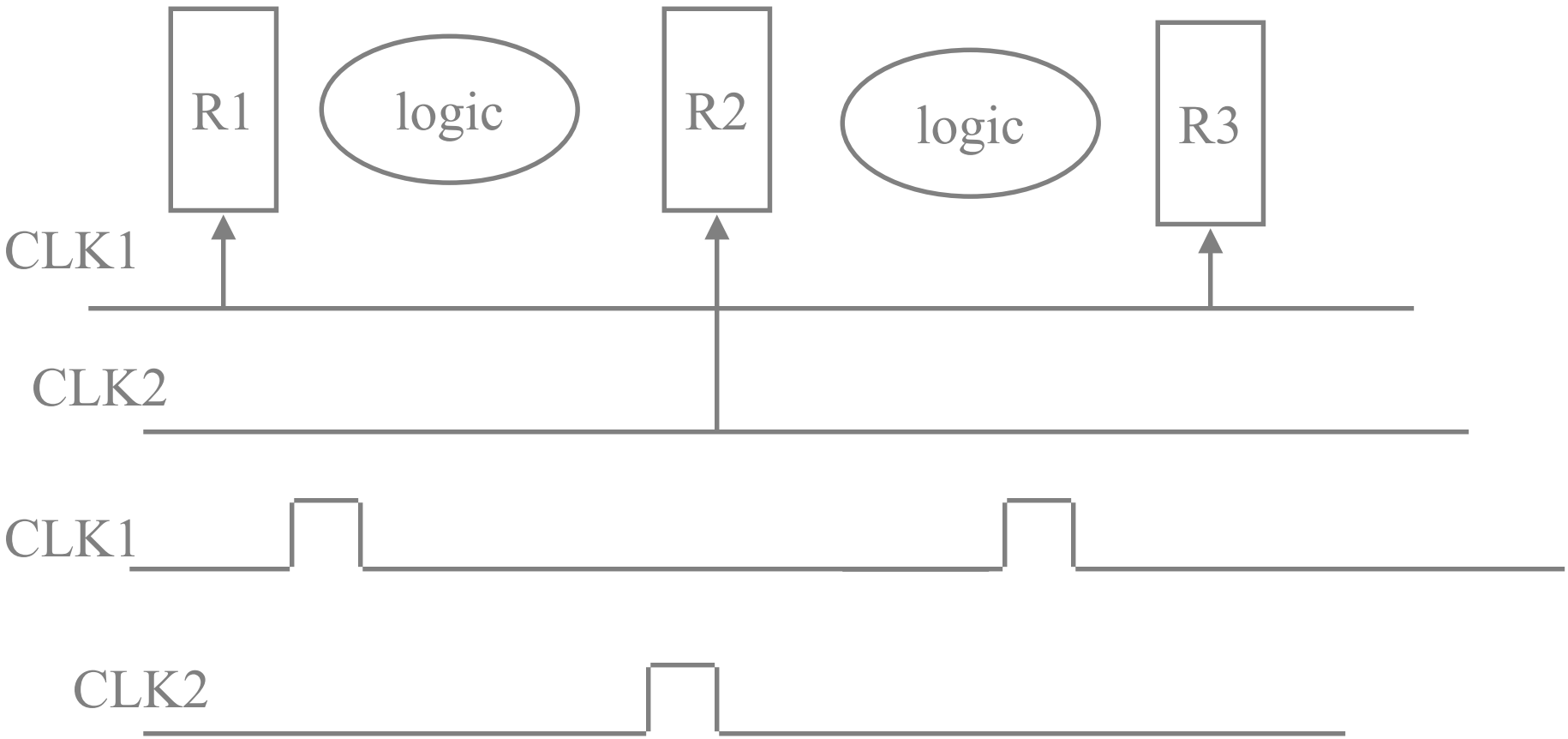
Note: Parameter values vary with technology and implementation.
E.g., set-up time t_g will generally be less for level-trigger than edge-trigger latch in same technology

Reliable Clocking

- **t_w > minimum pulse width and t_w > hold time**
- **$\Delta t > P_{max}$ + clock overhead**
- **$t_w < P_{min}$ for transparent latches**
  — can be avoided by
    - edge triggered dual rank registers
    - multiphase clock

Multiphase Clock

- Alternate stages use different clock phases
- clock phases don't overlap



Latching Summary

- **Edge-Triggered, Single-Rank**
  + Relatively simple to generate and distribute single clock
  -- Hazard for fast paths if P_{min} < clock skew
  but easy, inexpensive to pad gates for very short paths
  -- Cannot borrow time across latches
- **Edge-Triggered, Dual-Rank**
  + Safest, hazard-free clock
  -- Biggest clock overhead
- **Level-Triggered, Single Rank (*Pulsed Latch*)**
  + Minimum clock overhead
  + Few, simple latches => reduces area and power
  -- Hazard for fast paths
  -- Difficult to distribute, control narrow pulses
- **Level-Triggered, Dual-Rank**
  + Relatively simple to generate and distribute clock
  + Simple to avoid hazards with non-overlapped phases
  + Can borrow time across latches
  -- Larger clock overhead than single-rank

Skew

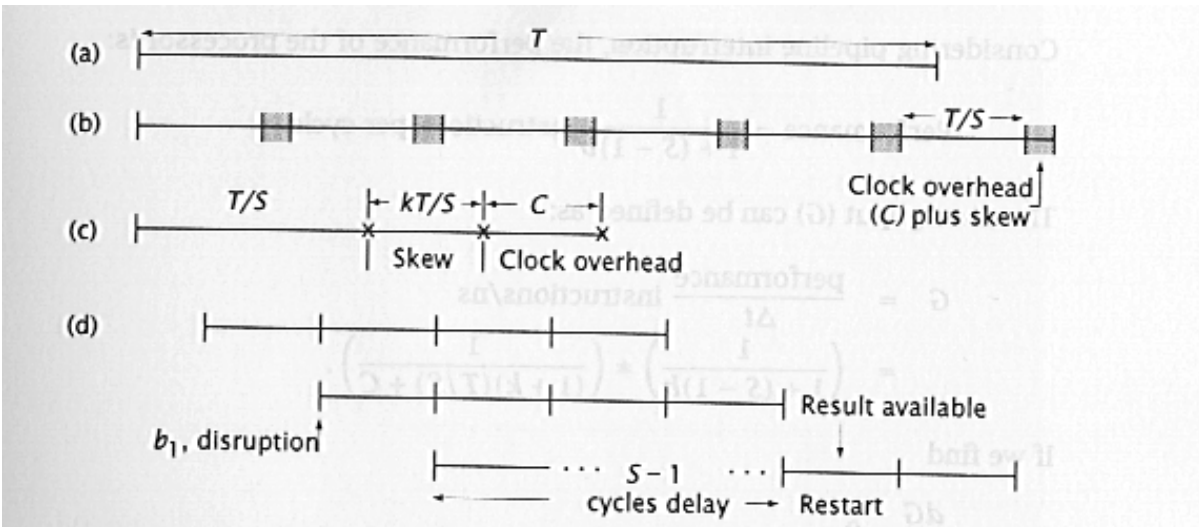
- **Skew is uncertainty in the clock arrival time**
- **two types of skew**
  - — depends on Δt.....skew = k, a fraction of P_{max} where P_{max} is the segment delay that determines Δt
    - large segments may have longer delay and skew
    - Part of skew varies with L_{eff}, like segment delay
  - — independent of Δt....skew = δ
    - Can relate to clock routing, jitter from environmental conditions, other effects unrelated to segment delay
- **effect of skew = $k(P_{max}) + \delta$**
  - — skew range adds directly to the clock overhead

Clocking Summary

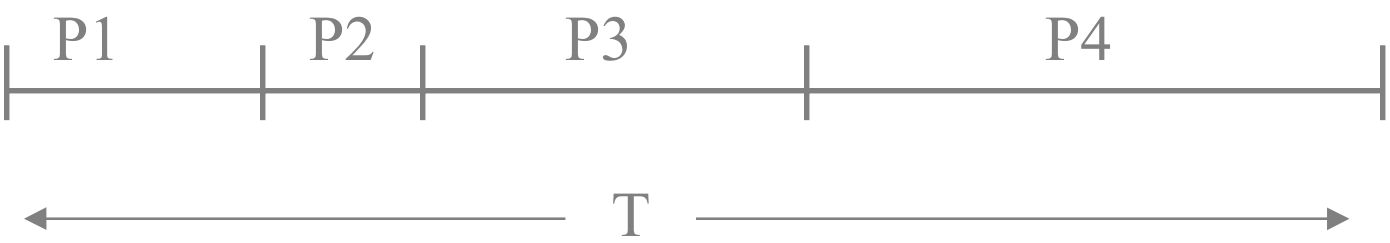
- **Overhead depends on clocking scheme and latch implementation**
  - Growing importance for microprocessors at frequencies > 300 MHz
  - Tradeoffs must be made carefully considering circuit, microarchitecture, CAD, system
  - Common approach
    - Distribute single clock to all blocks in balanced H-Tree
    - Gate clock at each block for power savings
    - Generate multiphase clocks for local circuit timing
  - Other approaches
    - Distribute single clock, but do not gate
      Use clock for both phases with TSPC latch
    - Distribute single clock, generate pulses locally for pulse latches (?)
  - Resulting parameter C is used in pipeline tradeoffs
- **Clock Skew has 2 components**
  - Variable component, factor k
    - We will use this to stretch Pmax
  - Constant (worst-case) factor δ
    - We will fold this into clock overhead C
- **And we have not even touched the issue of *asynchronous* design**

Optimum Pipelining

- Let the total instruction execution without pipelining and associated clock overhead be T
- In a pipelined processor let S be the number of segments in the pipeline
- Then S - 1 is the maximum number of cycles lost due to a pipeline break
- Let b = probability of a break
- Let C = clock overhead including fixed clock skew



Optimum Pipelining



$P_{max\ i}$ = delay of the i th functional unit

suppose $T = \Sigma_i P_{max\ i}$; without clock overhead

S = number of pipeline segments

C = clock overhead

$T/S >= \max (P_{max\ i})$ [quantization]

$$\Delta t = T/S + kT/S + C = (1+k)T/S + C$$

$$\text{Performance} = 1/(1+(S-1)b) \text{ [IPC]}$$

$$\text{Thruput} = G = \text{Performance} / \Delta t \text{ [IPS]}$$

$$G = (1 / (1+(S-1)b) \times (1 / ((1+k)(T/S)) + C)$$

Finding S for

$$dG/dS = 0$$

We get S_{opt}

Optimum Pipelining

$$Sopt = \sqrt{\frac{(1-b)(1+k)T}{bC}}$$

Finding S_{opt}

- **Estimate b and k....use k = 0.05 if unknown**
  - — b from instruction traces
- **Find T and C from design details**
  - — feasibility studies
- **Find S_{opt}**
- **Example**

b	k	T (ns)	C (ns)		Sopt	G (MIPS)	f (MHZ)	CPI	Clock Overhead %	
0.1	0.05	15	0.5		16.8	270	697	2.58	34.8%	
0.1	0.05	15	1		11.9	206	431	2.09	43.1%	
0.2	0.05	15	0.5		11.2	173	525	3.04	26.3%	
0.2	0.05	15	1		7.9	140	335	2.39	33.5%	

Clock Overhead = $C/\Delta T$

Quantization and Other Considerations

- **Now, consider the quantization effects**
  - T cannot be arbitrarily divided into segments
  - segments are defined by functional unit delays
  - some segments cannot be divided; others can be divided only at particular boundaries
- **some functional ops are atomic**
- **(usually) can't have cycle fractionally cross a function unit boundary**
- **S_{opt} ignores cost (area) of extra pipeline stages**
- **the above create quantization loss**
- **therefore: S_{opt} is the largest S to be used**
  - and the smallest cycle to be considered is $\Delta t = (1+k)T/S_{opt} + C$

Quantization

t_i = execution time of i^{th} unit or block
T = total instruction execution time w/o pipeline
S = no. pipeline stages
C = clock overhead
$t_m = \Delta t - C$ = time per stage for logic

$T = \Sigma_i\ t_i$ = time for instruction execution w/o pipeline

$S\Delta t = S(t_m + C)$ = (ignore variable skew)

$S\Delta t - T = S(t_m + C) - T$ (pipeline length overhead)
$= [St_m - \Sigma_i\ t_i] + SC$ [quantization overhead + clock overhead]

Vary # pipe stages => opposing effects of quantization/clock overhead

See Study 2.2 page 78

Microprocessor Design Practice (Part I)

- **Need to consider variation of b with S**
  - — Increasing S results in additional and longer pipe delays
- **Start design target at maximum frequency for ALU+bypass in single cycle**
  - — Critical to keep ALU+bypass in single clock for performance on general integer code
- **Tune frequency to minimize load delay through cache**
- **Try to fit rest of logic into pipeline at target frequency**
  - — Simplify critical paths, sacrificing IPC modestly if necessary
  - — Optimize paths with slack time to save area, power, effort

Microprocessor Design Practice (Part II)

- **Tradeoff around this design target**
  - — Optimal in-order integer pipe for RISC has 5-10 stages
    - Performance tradeoff is relatively flat across this range
    - Deeper for out-of-order or complex ISA (like Intel Architecture)
  - — Use longer pipeline (higher frequency) if
    - FP/multimedia vector performance are important and
    - clock overhead is low
  - — Else use shorter pipeline
    - especially if area/power/effort are critical to market success

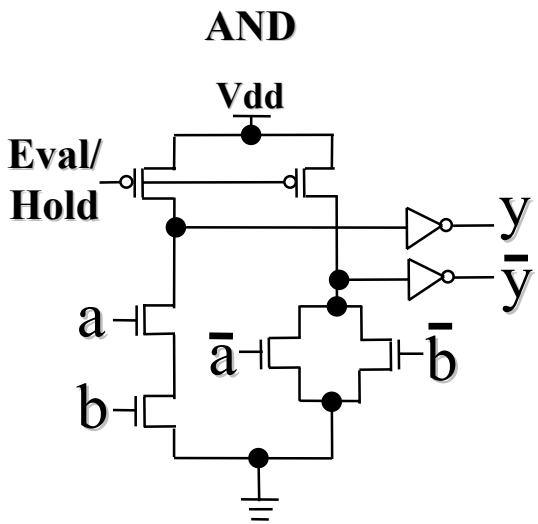
Advanced techniques

- **Asynchronous or self timed clocking**
  — avoids clock distribution problems but has its own overhead.
- **Multi phase domino clocking**
  — skew tolerant and low clock overhead; lots of power required and extra area.
- **Wave pipelining**
  — avoids clock overhead problems, but is sensitive to skew and hence clock distribution.

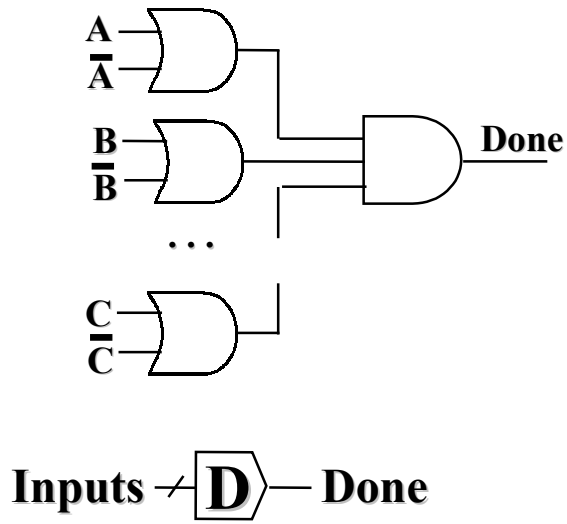
Self-Timed Circuits

Dual-Rail Logic Gate

Logic Value	
Reset	00
False	01
True	10
Invalid	11



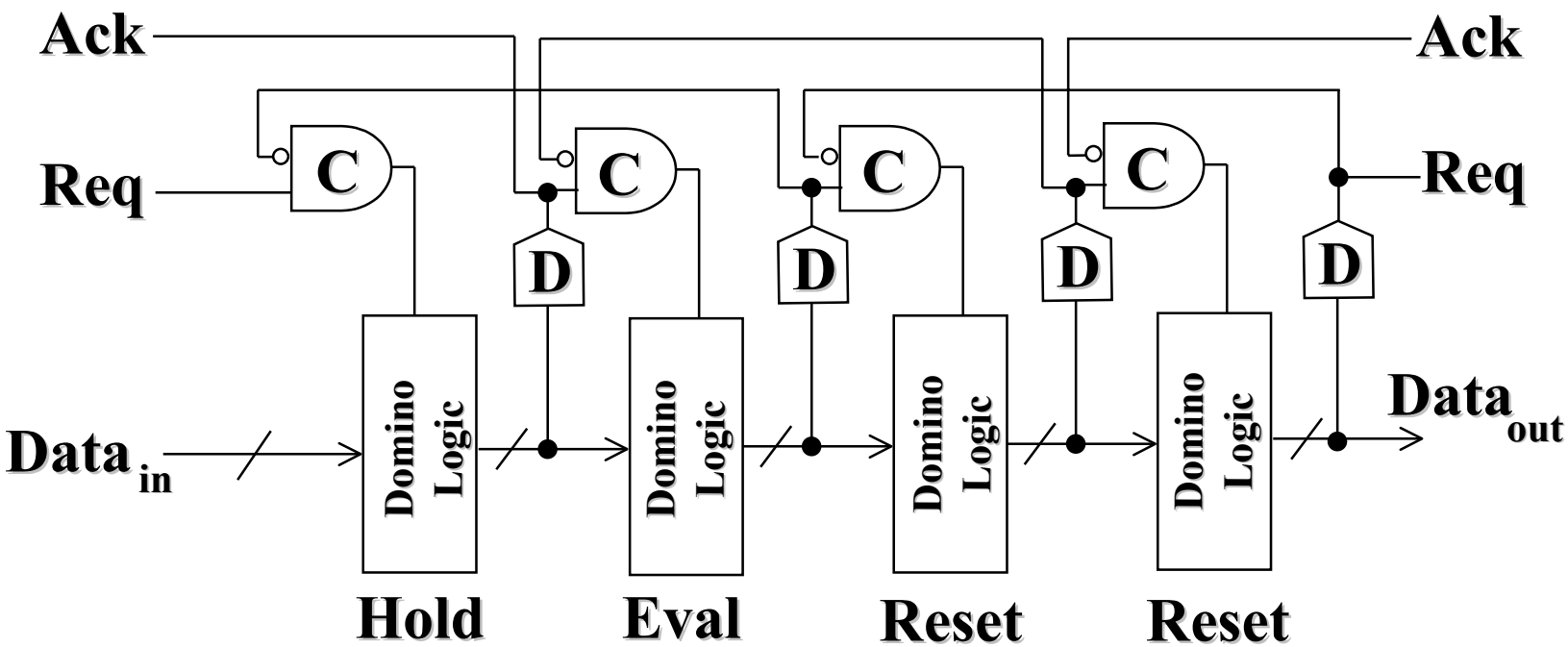
Completion Detection



1 J. Rabaey, Digital Integrated Circuits a Design Perspective, Prentice Hall 1996 ch. 9.

2 T. Williams and M. Horowitz, "A zero-overhead self-timed 160nS 54-b CMOS divider," IEEE Journal of Solid-State Circuits, vol. 26, pp.1651-1661, Nov. 1991.

Self-Timed Pipeline



Evaluation process

- C output is high for eval/hold; low for reset
- previous stage submits data; then req for eval(uation)
- D(one) signal is asserted when data inputs are available. This causes evaluation in this stage if its successor stage has been reset and its “D” signal is low.
- Overhead includes “D” and “C” logic and two segments of reset (precharge).

Self-Timed Circuit Summary

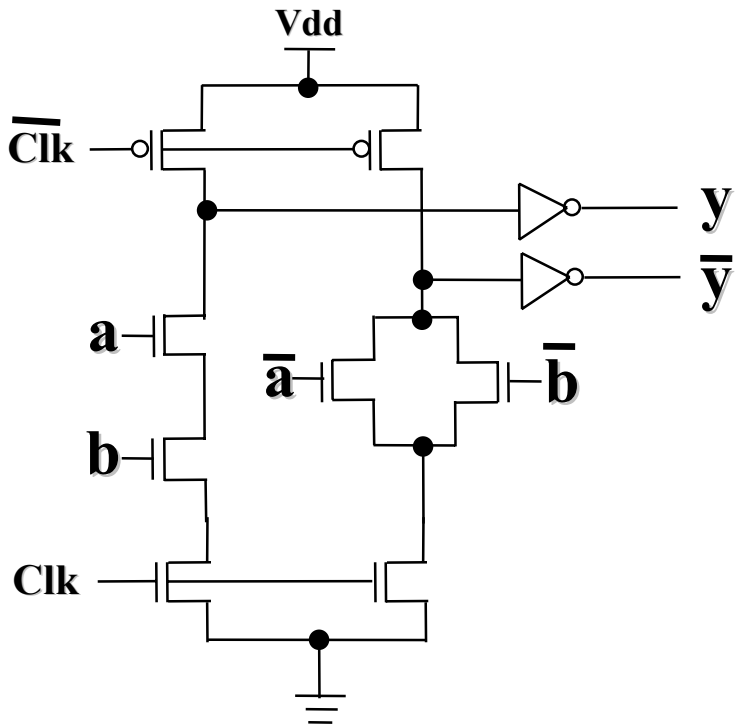
- **Delay-Insensitive Technique (Both gate and propagation delay)**
- **Can use fast Domino Logic**
- **Dual-rail logic implementation requires more Area**
- **Significant Overhead on Cycle time.**

Multi-Phase Domino Clock Techniques

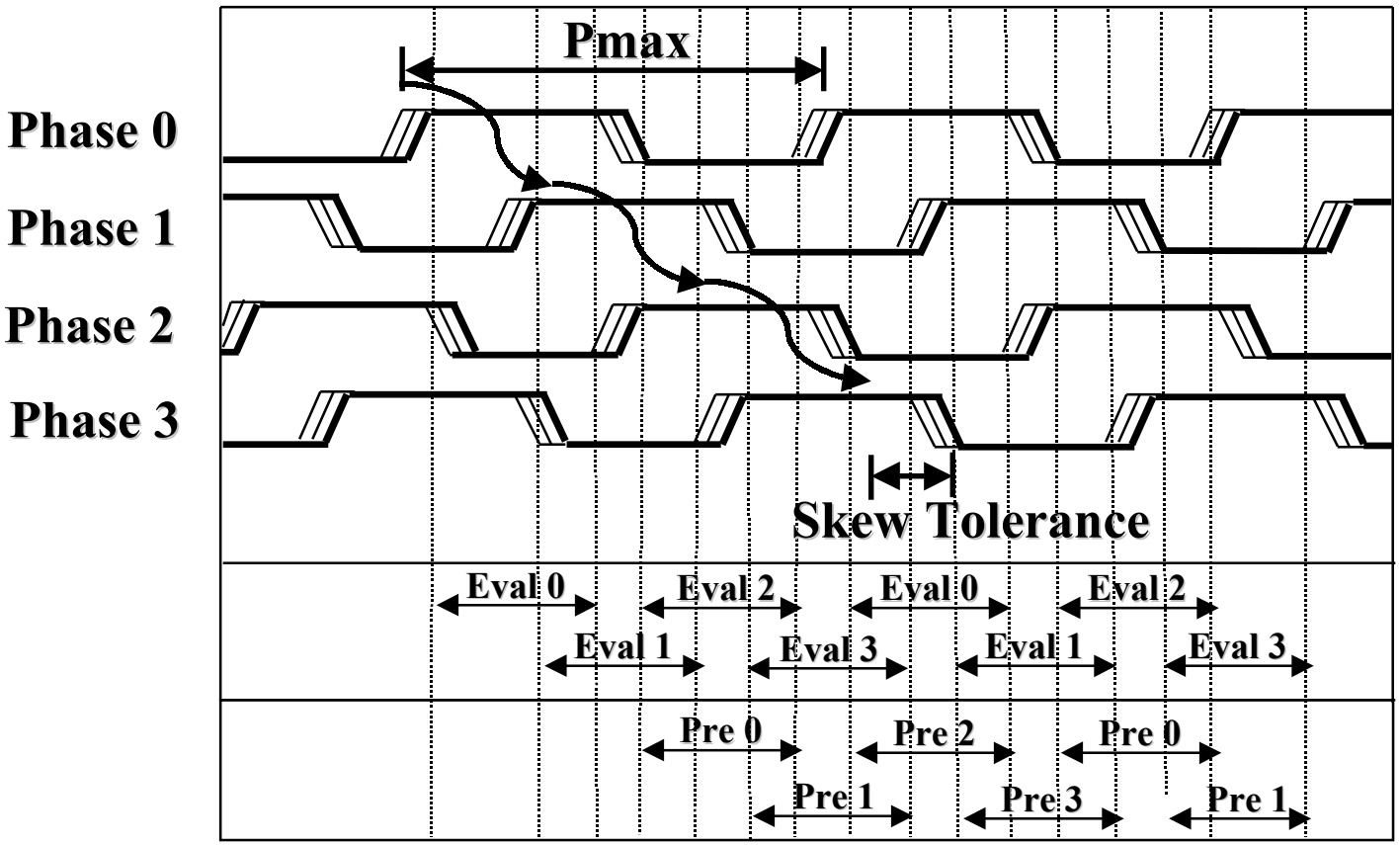
- Uses Domino Logic for Data Storage and Logical functions
- Reduces Clocking Overhead (Clock Skew, Latch Setup and Hold, Time Stealing)

- D. Harris and M. Horowitz, "Skew-Tolerant Domino Circuits," IEEE Journal of Solid-State Circuits, vol. 32, pp. 1702-1711, Nov. 1997.

Domino Logic AND Gate



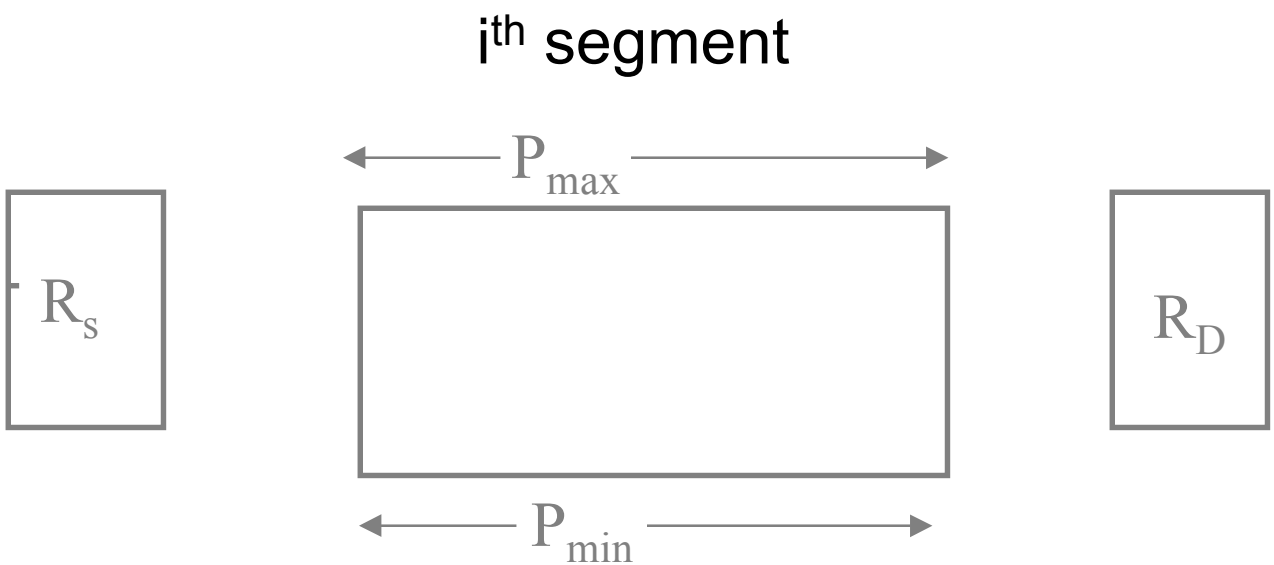
4-Phase Overlapped Clock



Wave Pipelining

- The ultimate limit on Δt
- Uses P_{min} as storage instead of latches.

Wave Pipelining



At time = t_1 let $data_1$ proceed into pipeline stage

It can be safely clocked at the destination latch at time t_3

$$t_3 = t_1 + P_{max} + C$$

But new $data_2$ can proceed into the pipeline earlier by an amount P_{min}

say, at time t_2, where $t_2 = t_1 + P_{max} + C - P_{min}$ so that

$$t_2 - t_1 = P_{max} - P_{min} + C$$

$$= \Delta t$$

Δt = the minimum cycle time for this segment

$$\text{minimum system } \Delta t = \{ \max \Delta t_i \}$$

over all i segments

Note that $data_1$ still must be clocked into the destination at t_3

For wave pipelining to work properly, the clock must be constructively skewed so that the data wave and the clock arrive at the same time.

Let CSi = constructive clock skew for the i th pipeline stage

Then :

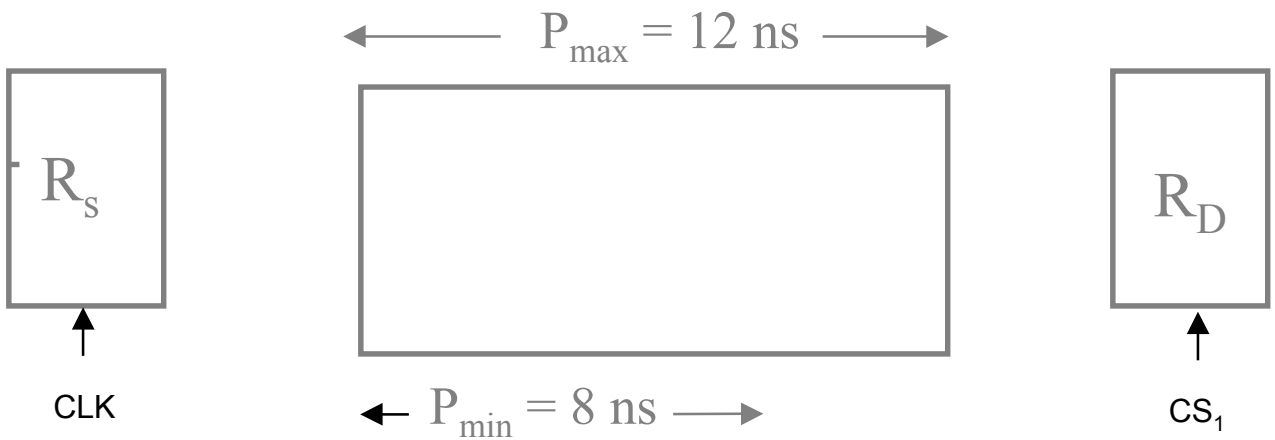
$CS_i = [\Sigma_{j=1} (P_{max} + C)_j]$ mod Δt, summed to the i th stage

the alternative is to force each stage to complete with the clock by adding delay, K, to both P_{max} and P_{min}, so that

$[\Sigma_{j=1} (P_{max} + K + C)_j]$ mod $\Delta t = 0$ for all i stages

since K is added to both P_{max} and $P_{min,}$ Δt is unaffected

Example



C= 1 ns

Wave and Optimum Pipelining

$$Sopt = \sqrt{\frac{(1-b)(1+k)T}{bC}}$$

b in the above also acts as a limit on the usefulness of wave pipelining, since only those applications with low b or large S can effectively use the low Δt available from wave pipelining. These applications would include vector and signal processors.

Limits on Wave Pipelining

The limit on the difference, $P_{max} - P_{min}$, has two components

$$\text{let } v = P_{max} - P_{min} = f(\alpha, \beta)$$

α is the static design variation and β is the environmental variance

typically $\alpha = P_{max} / P_{min}$ is controllable to 1.1. Ignoring C this allows 10 "waves" of data in a pipeline. But usually β is a more constraining limit. Unless on-chip compensation (thru the power supply) is used the limit on $\beta = P_{max} / P_{min}$ is only 2 or even 3 limiting the improvement on Δt to 3 or 2 times the conventional Δt

Summary

- **Minimizing clock overhead is critical to high performance pipeline design**
- **Exploring limits for optimal pipelines can bound design space and give insight to tradeoff sensitivity**
- **Vector pipeline frequency is limited by variability in delay, not max delay**
  - Performance (throughput or frequency) improves as much from increasing minimum delay as from reducing max delay
  - Wave pipelining and similar techniques may prove practical
- **Rest of course will assume conventional clocking with cycle time set by max delay and clock skew**