**CLOCK SK EW**

Unequal delay in distribution of the clock signal
to various parts of a circuit.
- can lead to erroneous behavior
- Comes about because:
  - clock wires have delay
  - buffers have inequalities

\[
\text{clk} \quad \text{clk}\]
\[
\text{clk'} \quad \text{clk'}
\]

\[
\rightarrow \quad \text{delay in distribution}
\]

if clock period \( T = T_{\text{clk}} + T_{\text{setup}} + T_{\text{clk-to}} \)
circuit will fail!

Therefore:
1. Control clock skew
   - careful clock distribution
   - Equalize path delay on wires & buffers
   - don't gate clocks

2. \( T \geq T_{\text{clk}} + T_{\text{setup}} + T_{\text{clk-to}} + \text{worse case skew} \)

Modern large chips (processors) control end to
end chip skew to \( \sim 200 \)ps

This is getting harder all the time.
Pipelining Technique for improving the throughput of a piece of logic.

Analog to washing clothes:

\[
\begin{array}{c|c|c}
\text{step 1} & \text{wash} & (20 \text{ min}) \\
\text{step 2} & \text{dry} & (20 \text{ min}) \\
\text{step 3} & \text{fold} & (20 \text{ min}) \\
\hline
\text{per load} & \text{60 min}
\end{array}
\]

\[x \times 4 \text{ loads} \Rightarrow 4 \text{ hrs.}\]

Overlapped \(\Rightarrow 2 \text{ hrs.}\)

In the limit as we increase the # of loads the average time per load approaches 20 min.

The latency for one load = 60 min.

\(\Rightarrow\) time from start to end

The throughput = 3 loads/hr.

The pipelined throughput = 
\[
\# \text{ of pipe stages} \times \text{unpipelined throughput}
\]
Pipelining Hardware

Cut the block into pieces (stages) & separate with registers.

\[ T = 8 \text{ ns} \]
\[ T_{FF} (\text{setup} + \text{CLK} \rightarrow Q) = 1 \text{ ns} \]
\[ f = \frac{1}{9 \text{ ns}} = 111 \text{ MHz} \]

\[ T_1 = T_2 = 4 \text{ ns} \]
\[ T' = 4 \text{ ns} + 1 \text{ ns} + 4 \text{ ns} + 1 \text{ ns} = 10 \text{ ns} \]

\[ f = \frac{1}{9 \text{ ns} + 1 \text{ ns}} = 100 \text{ MHz} \]

CL logic block produces a new result every 4 ns instead of every 9 ns.
Limits on Pipelining

- With out FF overhead throughput improvement $\times$ # of stages
- After many stages are added, FF overhead begins to dominate.

```
throughput ($\frac{1}{T}$)
```

0
2
4
6
8

# of stages

- Unequal stages
- FFs dominate cost
- Clock power

Example use: $f(x) = y = a x_i^2 + b x_i + c$

Data flow graph:

3 (nearly) equal stage

Insert pipeline registers at dashed lines.
Example: How do we pipeline an adder?
Review

Any CL block can be pipelined to increase its throughput at the expense of pipeline registers (FFs).

Example:

1. Draw "cut" lines.
2. Where-ever cut line crosses a wire - add a FF.

Pipelining uses circuits more efficiently, because multiple sets of data can be processed simultaneously.

However, the sets of data must be independent, which brings us to...
Feedback & pipelining:

Example: \( y_i = ay_{i-1} + x_i + b \)

Try 3 deep pipeline

The only way:
- Send in: \( x_i, 0, 0, x_{i+1}, 0, 0, x_{i+2}, 0, \ldots \)
- Get out: \( y_i, 0, 0, y_{i+1}, 0, \ldots \)

How does performance compare?
- Latency = as unpipelined version
- Of course can't get benefit of pipelining

If we have 3 or more independent streams (channels):
- \( x_i, x_i, x_i, x_{i+1}, x_{i+1}, x_{i+1}, x_{i+2} \)
- \( y_i, y_i, y_i, \ldots \)

"C-slow" technique: C streams (problem instances) each running at \( \frac{1}{c} \) rate.
Typical homework (or exam) problem:
For the given circuit (unpipelined):

\[ \begin{align*}
&\text{\textbf{setup}} = 10 \text{ ns} \\
&\text{\textbf{adder}} = 12 \text{ ns} \\
&\text{\textbf{loff}} (\text{setup + clk\rightarrow Q + skew margin}) \\
&\quad = 2 \text{ ns}
\end{align*} \]

a) What is the unpipelined throughput for the circuit?  
b) Draw a new version with two pipeline stages: 

c) What is the new throughput?  
\"\"\" latency?
Solution:

a) \[
\frac{1}{10 \text{ns} + 12 \text{ns} + 2 \text{ns}} = \frac{1}{24 \text{ns}} = 42 \text{MHz}
\]

b)

\[
\begin{align*}
\text{reg} & \quad \text{reg} \\
\text{Shifter} & \quad \text{reg} \\
\text{reg} & \quad \text{reg} \\
\text{adder} & \quad \text{reg}
\end{align*}
\]

c) Throughput: \[
\frac{1}{12 \text{ns} + 2 \text{ns}} = \frac{1}{14 \text{ns}} = 71 \text{MHz}
\]

latency: \[
10 \text{ns} + 2 \text{ns} + 12 \text{ns} + 2 \text{ns} = 26 \text{ns}
\]