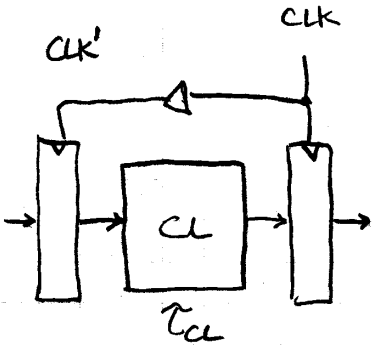
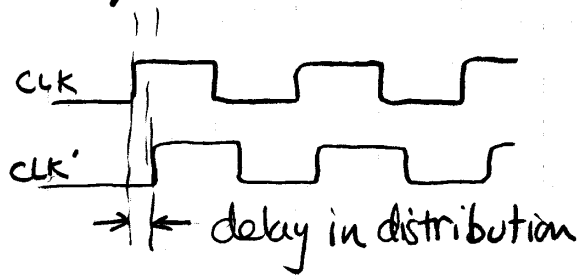
CLOCK SKEW

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





if clock period $T = \tau_{CL} + \tau_{setup} + \tau_{CLK \to Q}$
circuit will fail!

Therefore:

1. Control clock skew
   - careful clock distribution. equalize path delay on wires & buffers.
   - don't gate clocks.
2. $T \geq \tau_{CL} + \tau_{setup} + \tau_{CLK \to Q}$ + worse case skew

Modern large chips (uprocessors) control end to end chip skew to ~ 200ps.

This is getting harder all the time.

<u>Pipelining</u> Technique for improving the throughput of a piece of logic.

Analog to washing clothes:

per load	step 1	wash	(20 min)
	step 2	dry	(20 min)
	step 3	fold	(20 min)
			60 min

x 4 loads => 4 hrs.

← 20 min →

wash	load1	load2	load3	load4		
dry		load1	load2	load3	load4	
fold			load1	load2	load3	load4

Overlapped => 2 hrs.

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

The <u>latency</u> for one load = 60 min.

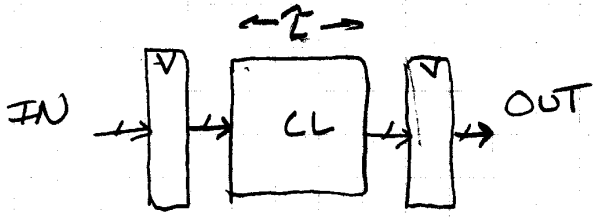
↳ time from start to end

The <u>throughput</u> = 3 loads/hr.

The pipelined throughput ≈

\# of pipe stages * unpipelined throughput.

Pipelining Hardware

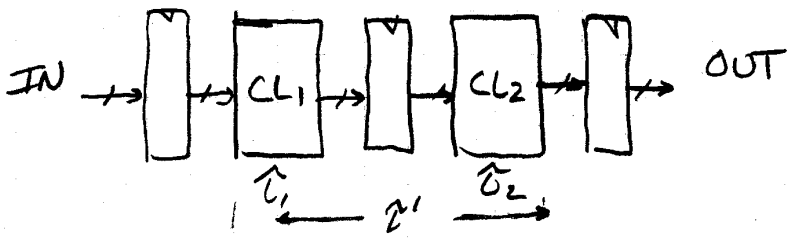


assume $\tau = 8\,ns$

τ_{FF} (setup + CLK→Q) = 1ns

$f = \frac{1}{9ns} = 111\,MHz$

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



assume $\tau_1 = \tau_2 = 4ns$

$\tau_1 = \tau_2 = 4ns$

$\tau' = 4ns + 1ns + 4ns + 1ns = 10ns$

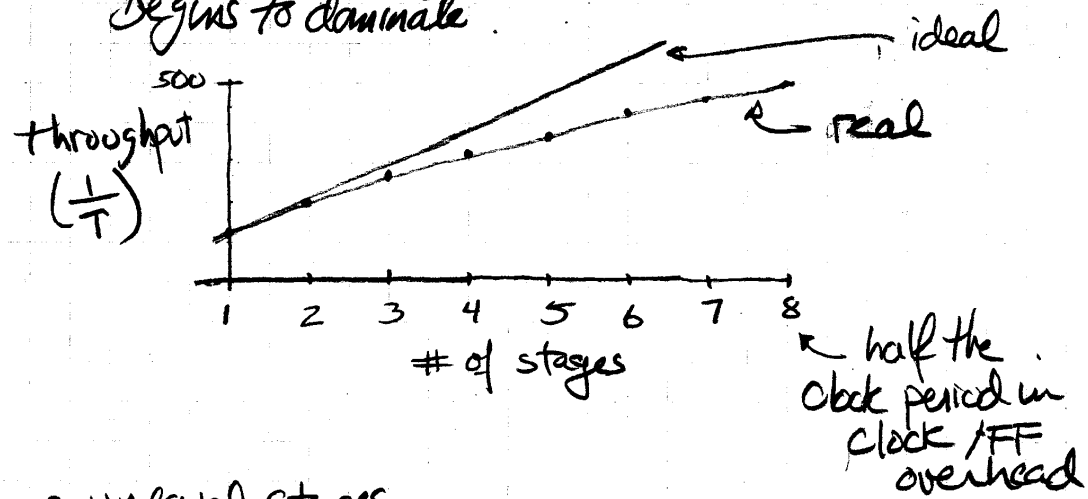
$f = \frac{1}{4n + 1ns} = 200\,MHz$

CL logic block produces a new result every 5ns instead of every 9ns.

Limits on Pipelining

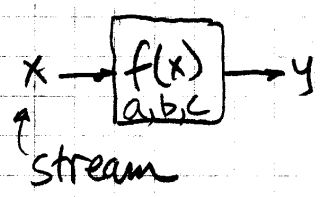
②

- without FF overhead throughput improvement $\propto$ # of stages
- after many stages are added, FF overhead begins to dominate.

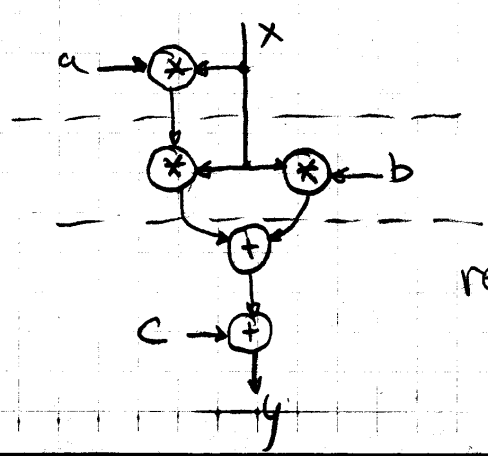


- unequal stages
- FFs dominate cost
- clock power

Example use: $f(x) = y_i = ax_i^2 + bx_i + c$



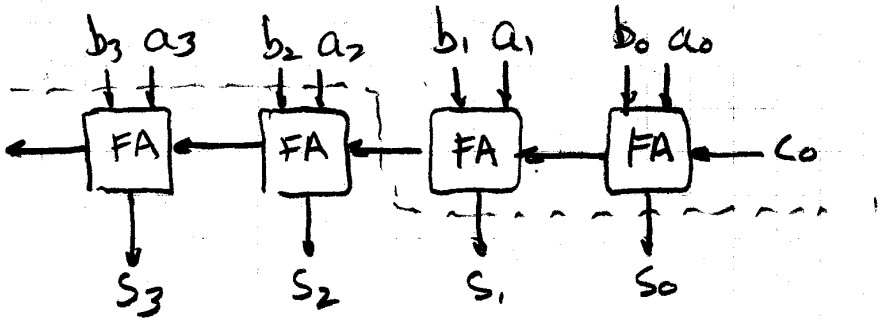
Data-flow graph:

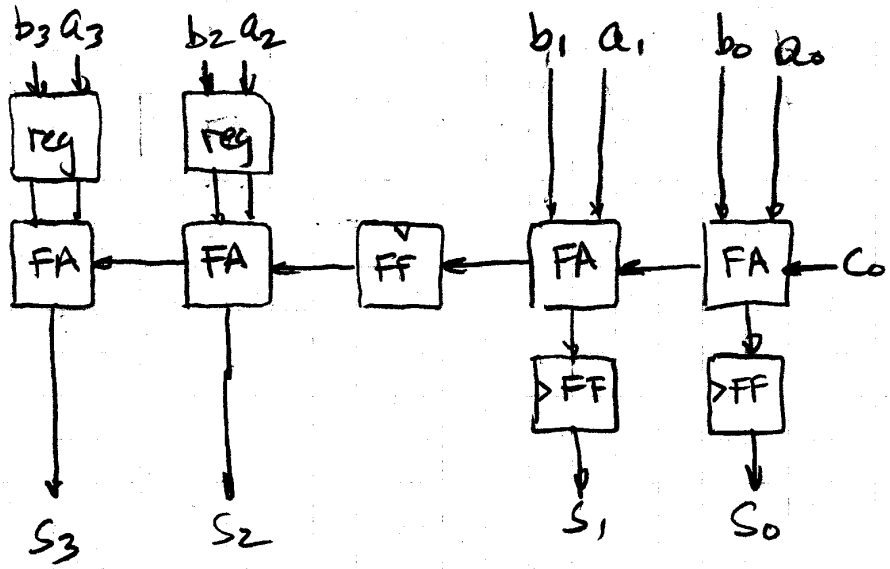


3 (nearly) equal stage

Insert pipeline registers at dashed lines.

Example: How do we pipeline an adder?

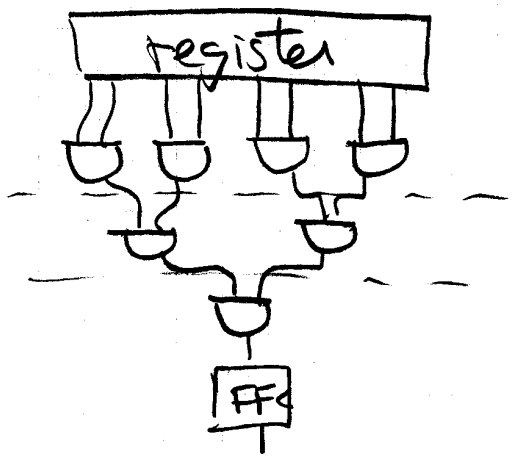




Review

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

ex:



'AND' reduction

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

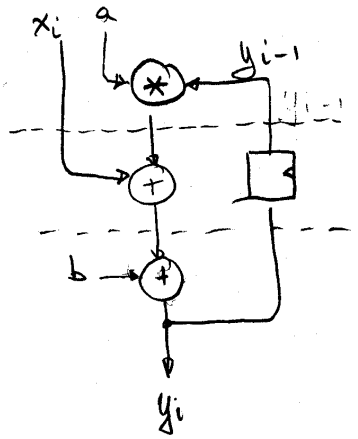
Pipelining uses circuits more efficiently, because multiple sets of data can be processed simultaneously.

However, the sets of data must be independent. which brings us to...

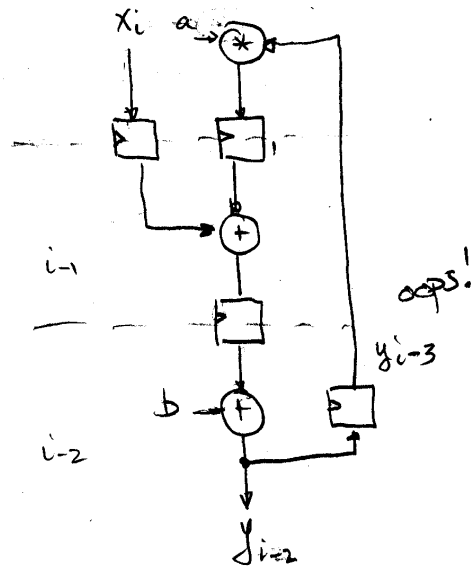
Feedback & pipelining:

5

example $y_i = a y_{i-1} + x_i + b$



try 3 deep pipeline



The only way:

send in $x_i, \emptyset, \emptyset, x_{i+1}, \emptyset, \emptyset, x_{i+2}, \emptyset, \ldots$

get out $y_i, \emptyset, \emptyset, y_{i+1}, \emptyset, \ldots$

How does performance compare?

Latency ≈ as unpiplined version

Off course don't get benefit of pipelining

If we have 3 or more independent streams (channels):

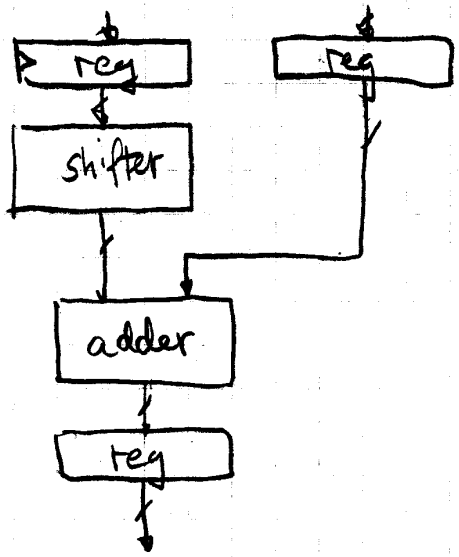
$x_i^0, x_i^1, x_i^2, x_{i+1}^0, x_{i+1}^1, x_{i+1}^2, x_{i+2}^0$

$y_i^0, y_i^1, y_i^2, \ldots$

"C-slow" technique – C streams (problem instances) each running at $\frac{1}{C}$ rate.

Typical homework (or exam) problem:

For the given circuit (unpipelined):



$\tau_{shifter} = 10$ ns

$\tau_{adder} = 12$ ns

τ_{FF} (setup + clk→Q + skew margin) $= 2$ ns

a) What is the <u>unpipelined</u> throughput for the circuit?

b) Draw a new version with two pipeline stages:

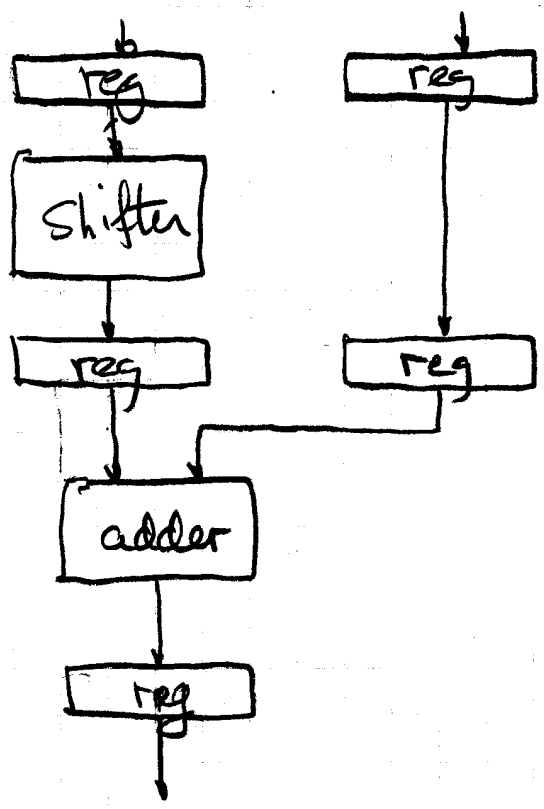
c) what is the new throughput?

" " " " latency?

Solution:

a) $$\frac{1}{10\,ns + 12\,ns + 2\,ns} = \frac{1}{24\,ns} = 42\,MHz$$

b)



c) Throughput : $$\frac{1}{12\,ns + 2\,ns} = \frac{1}{14\,ns} = 71\,MHz$$

latency : $10\,ns + 2\,ns + 12\,ns + 2\,ns = 26\,ns$