What sets the clock frequency

Everyone knows that a chip has a clock speed, but few people are told what sets it. The answer is in the kind of layout you have been drawing, and this page works it out from numbers you measure in Lab 05 and the capstone.

The timing rule

Between two flip-flops there is some logic. In one clock period, three things have to happen in sequence:

1. the first flip-flop has to produce its output after the clock edge: the **clock-to-Q** delay
2. that value has to get through the logic and the wires: **the path delay**
3. it has to be stable at the second flip-flop's input a little *before* the next clock edge: the **setup time**

$$T_{\text{clk}} \geq t_{\text{clk}\to Q} + t_{\text{logic}} + t_{\text{setup}}$$

$$f_{\max} = \frac{1}{T_{\text{clk}}}$$

Static timing analysis is built on this one rule, and the rest of it is bookkeeping around those three terms.

One gate delay

The fastest thing this process can do is one inverter driving one inverter and nothing else, and you measured exactly that in Lab 05 with a **ring oscillator**, an odd number of inverters connected in a loop. Lab 05, Part 5 explains why such a loop can't settle and oscillates instead, with a period of $2Nt_p$ for N stages that each take t_p. The lab builds two 31-stage rings out of extracted layouts, one from the foundry's inverter and one from yours, and measures them:

	period	frequency	one stage
the foundry's `inv_1`	2033.6 ps	491.7 MHz	**32.80 ps**
your Lab 03 inverter	2640.2 ps	378.8 MHz	**42.58 ps**

So **33 picoseconds** is the floor: the delay of one gate in SKY130 at 1.8 V driving nothing but a copy of itself. Every other delay on this page can be counted in multiples of it.

A real path

The capstone block is a flip-flop, two AND gates and a buffer, placed in a row 14.6 µm long. If you extract it with parasitics, apply a clock edge and watch Y, you get these numbers:

```
CLK rise -> Y rise        536.17 ps    (measured on your extracted layout)
D setup at the flop       103.32 ps    (sky130_fd_sc_hd__tt_025C_1v80.lib)
shortest legal period     639.49 ps
f_max                    1563.75 MHz
```

That works out to **1.56 GHz**, for four levels of logic and almost no wire.

The two delays in it come from two different places:

- **The 536 ps is simulated** on your own geometry, with every parasitic capacitance Magic could find.
- **The 103 ps is looked up** in `sky130_fd_sc_hd__tt_025C_1v80.lib`, the same Liberty file the digital track synthesises against in DD103. Setup time depends on the flip-flop's internals, so you can't get it from your own layout. This is the point where the analog and digital tracks read the same file.

536 ps is **16 inverter delays** for only four gates, so it isn't the number of gates that makes the path slow. Most of it is the flip-flop's clock-to-Q, and the two `and2_0` cells you are given are the weakest AND in the library.

Why a real chip runs at 200 MHz

The gap between your 1.56 GHz and a real chip's 200 MHz comes out of the same arithmetic, and you can follow it one term at a time.

Logic depth. Four levels of logic is a toy. A real critical path, such as a 32-bit adder carry chain, a cache tag comparison, or a decoder feeding a mux tree, is routinely twenty to forty levels deep. At the same cost per level, about 134 ps (536 ps over four levels), that comes to 2.7–5.4 ns before any of the terms below.

Wires. Your capstone's longest net is about 2 µm. A net crossing a modest block is hundreds of microns long, and a bus crossing a die is millimetres. Wire delay grows roughly with the *square* of length, because both resistance and capacitance grow with it. Past a few hundred microns, almost all of the delay is in the wire, and the gates are a rounding error.

DD103′s ripple-adder page measures the same effect in the digital track: a 4-bit ripple carry takes 1.481 ns and a 64-bit one takes 24.996 ns, so the delay comes from the length of the *chain* rather than from any one gate.

Clock skew and jitter. The clock edge doesn't arrive at every flip-flop at the same moment, and every picosecond of disagreement comes straight off the time budget.

Margin. All of the numbers above are for the typical corner at 25 °C. Sign-off is done at the slow-slow corner, at 125 °C and low supply, where everything is 30–50 % worse, and the chip still has to work there.

If you stack all of those on the 1.56 GHz you measured, a few hundred megahertz is what is left. Nobody has to be careless for that to happen; it's what the arithmetic gives.

The trade-offs

You can attack every one of those terms, and every attack costs something you also care about:

You want	You do	You pay
less path delay	use stronger cells (`and2_2` not `and2_0`)	area, and input capacitance upstream
less path delay	fewer levels of logic	more gates in parallel, more area
less wire delay	put connected things next to each other	the work of placement, which is a global problem
less self-load	pack the layout tighter	an afternoon of your time (Lab 05 measures what it buys)
more headroom	pipeline it: cut the path with another flip-flop	latency, and a flip-flop's worth of area and power

Lab 05 measures the fourth row directly: in a ring, your loose inverter is 30 % slower than the foundry's, even though the transistors are identical.

Because none of these moves is free, a synthesis and place-and-route tool has to choose among them, a few million times over, guided by an objective function. This course has been about understanding what it is choosing between.