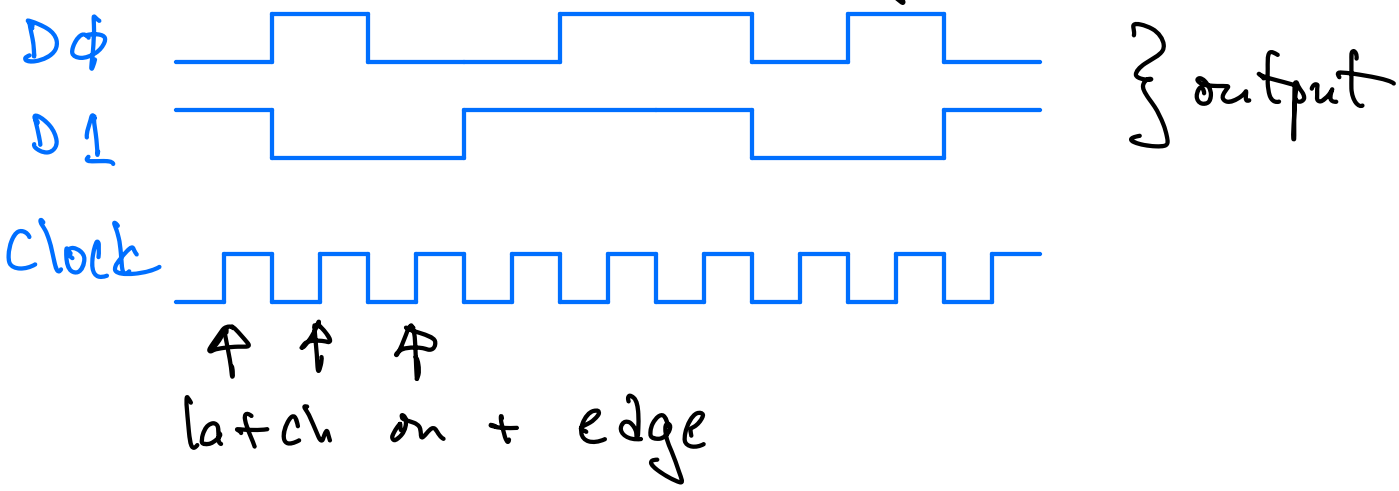
Parallel data bits gives most info per clock tick (need to decide latching on + or - clock edge - we will use +)



Receiver latches on + edge so should see:

D0: 0 1 0 0 1 1 0 1 0

D1: 1 0 0 1 1 1 0 0 1

This works ok for 2 bits.

Bandwidth: #bits * clock freq

ex: clock is 1MHz, BW = 2Mbps (bits/sec)

to get more throughput:

=> Add more bits

=> Increase clock freq

Problems:

1. add more bits => stochastic errors increase!

ex: let p = prob of "noise" flipping a bit
(transistor heats up, oscillations, etc)

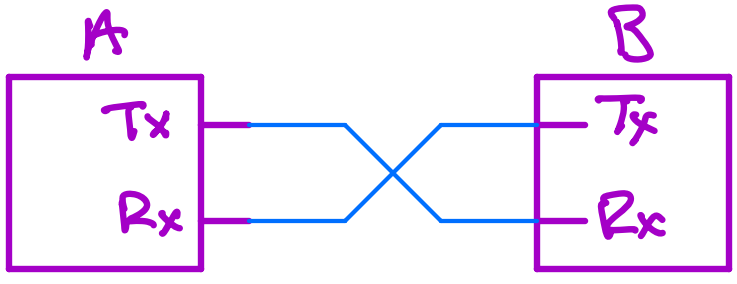
Prob of any 1 of N bits to flip $\propto Np$

so as $N \uparrow$, Prob of error $\uparrow$

2. increase clock lag
data transitions are not infinitely fast
(need to wait for transitions)
and subject to fluctuations

Solution: serial transmission, no clock (soft)
there are many variations
⇒ Universal Asynchronous Receiver Transmitter
(UART)

ex:



no clock!

Here's the protocol that A & B both follow:

1. assert Tx line (drive to high, or 1)
2. watch their Rx line

3. agreement in advance on width (in time) between transitions

⇒ call it ΔT, then bit freq is $f = \frac{1}{\Delta T}$ (baud rate)

4. agreement on number data bits to send for 1 cycle (usually 1 byte = 8 bits data)

Transmitter:

- deassert Tx to start transmission
- wait ΔT - 1st bit is called "START"
- drive Tx high to send a 1, low for 0
- send N bits of data like this (usually N = 8 for 1 byte)
- assert Tx for ΔT to end transmission

So 1 cycle = 1 start, 8 data, 1 stop bit

ex: baud rate = 9600 b/sec

byte rate = 960 B/sec

optional: add a "parity" bit before stop bit

parity = 0 if #1st is even,

1 if odd

this can be useful if you have a non negligible single bit error rate, but small enough so that 2 bits in error is rare

parity P = xor of all data bits

Receiver:

- watch Rx, wait for transition to 0
- wait ΔT, then latch Rx in the middle of next N ΔT intervals
- latch parity bit if agreement in advance
- confirm stop bit, present N bits of info
- back to wait