

CS2002 Logic Lecture 7

More Sequential Circuits

Ian Gent

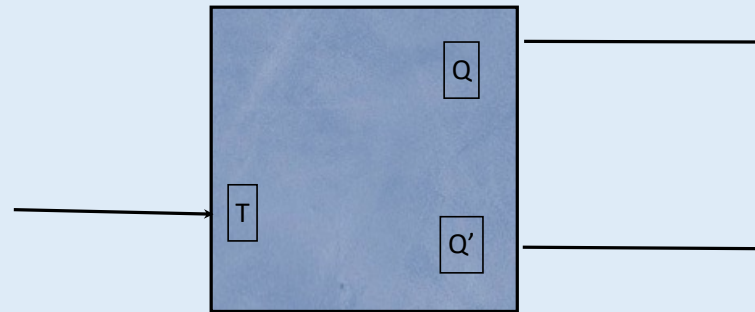
Last Time

- Sequential Logic Circuits
- Interlude: LogicSim for simulating circuits
- CPU Clocks

This Time

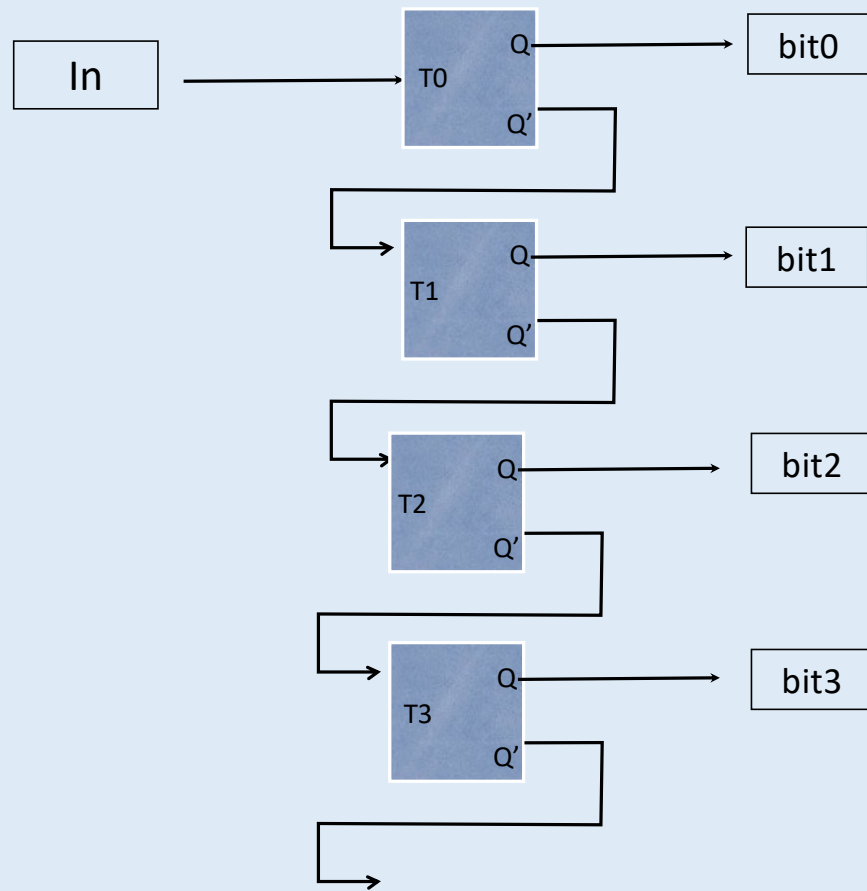
- Flip Flops
- More Combinational Circuits
- Then introduction to Architecture

Edge Triggered T Flip-Flop

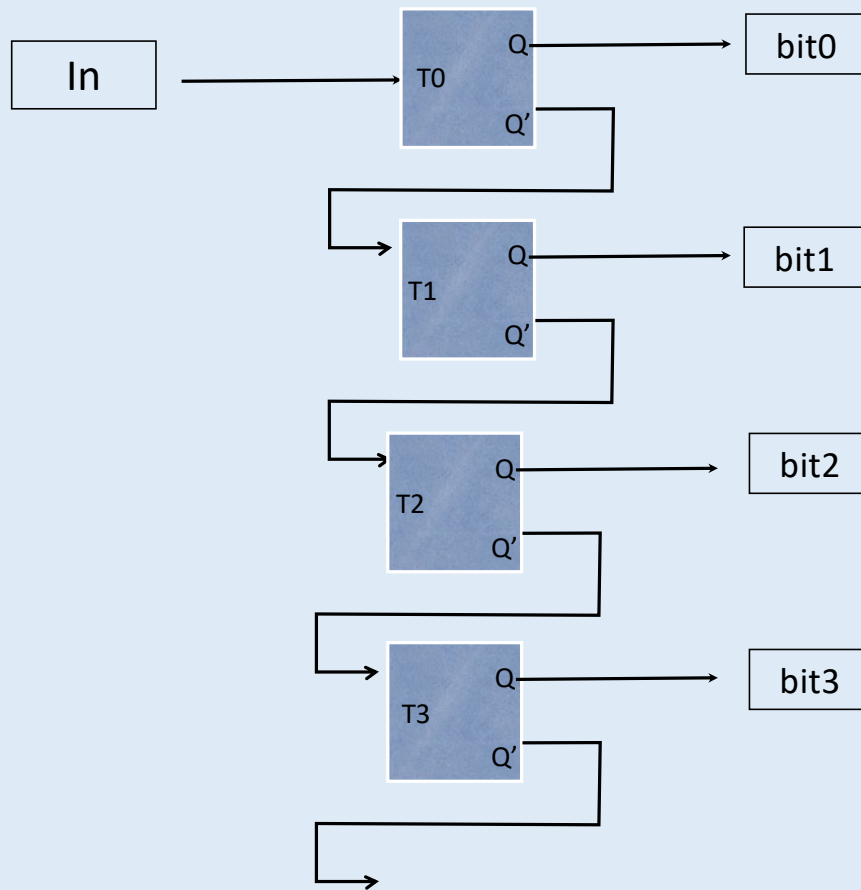


- When T goes from low to high Q and Q' change over
- Halves clock speed
- Can be built with gates, but more easily directly out of transistors

Counting with Flip Flops

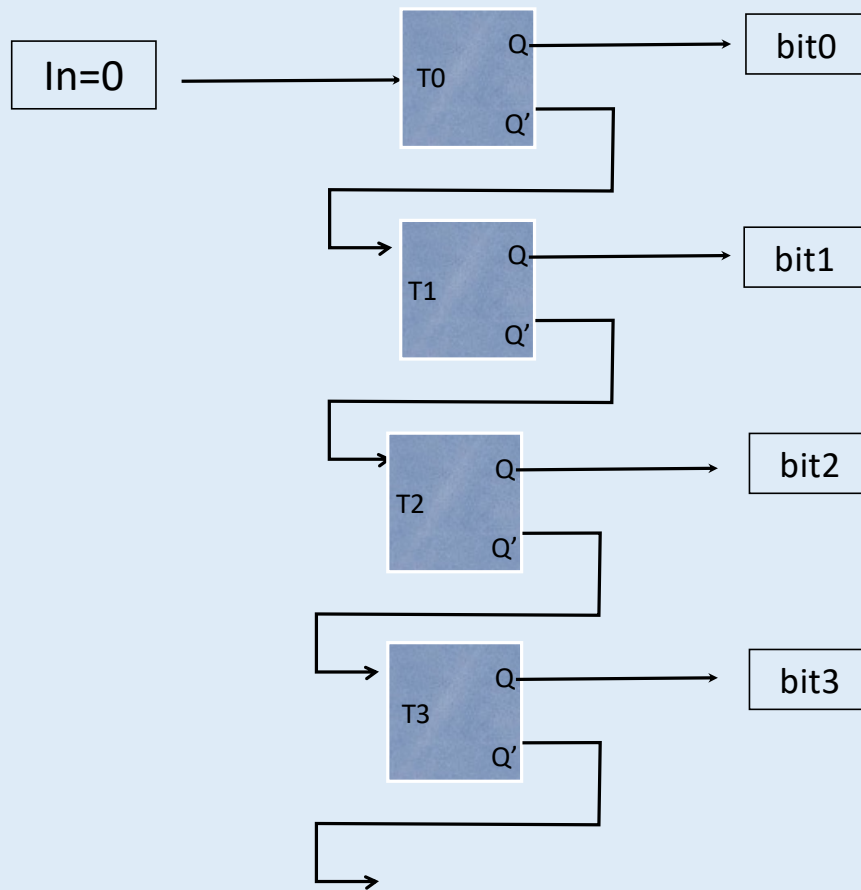


Counting with Flip Flops



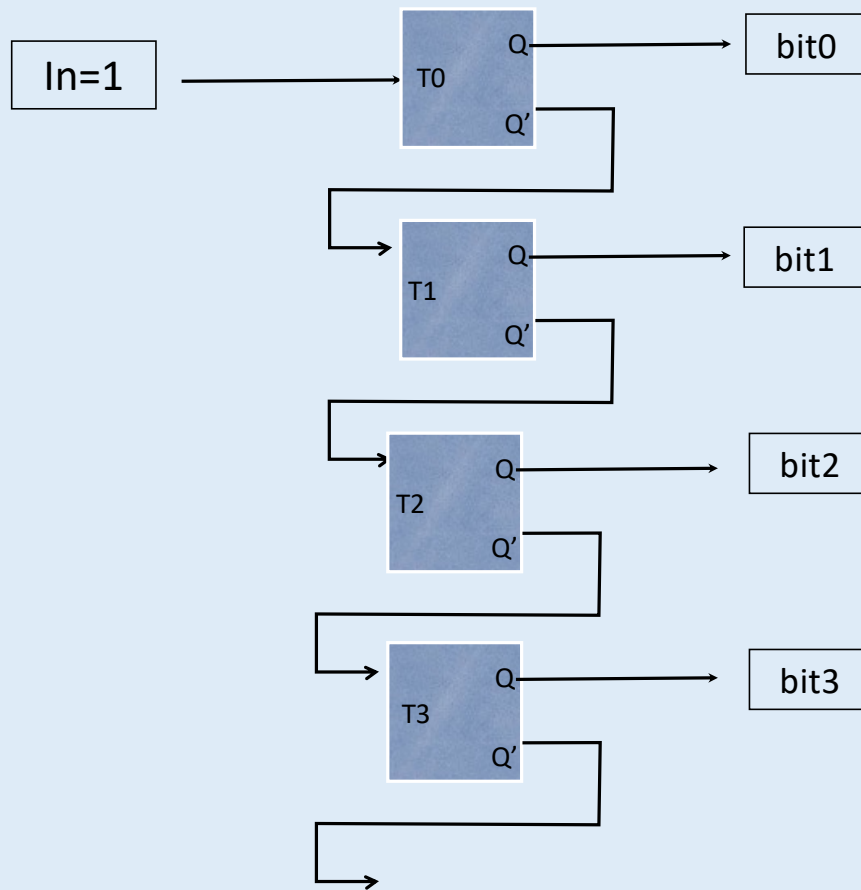
- This counts the number of rising edges at *In*
- i.e. the number of times *In* changes from 0 to 1
- As 4 bit number
 - *bit0* least significant
- How??

Counting with Flip Flops



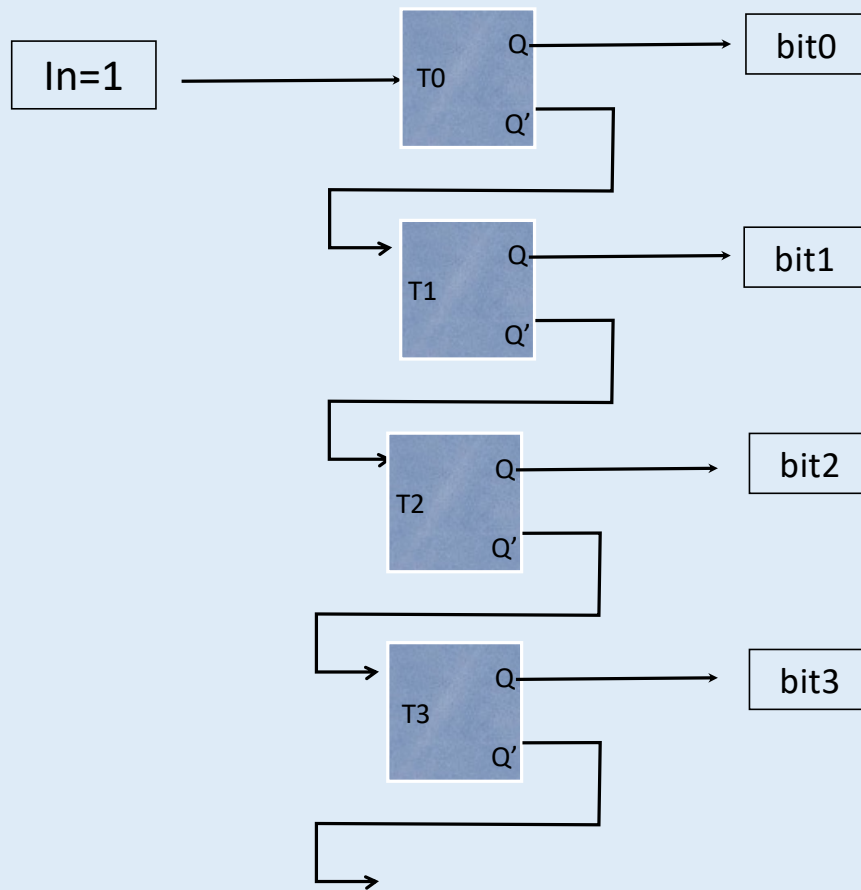
- What does this circuit do?
- Let's assume:
 - all Qs start as 0
 - all Q's start as 1
 - $In=0$
- So we have
 - $Q0 = 0$
 - $Q1 = 0$
 - $Q2 = 0$
 - $Q3 = 0$
- Reminder:
 - each T flip flop swaps Q and Q' when a rising edge comes in

Counting with Flip Flops



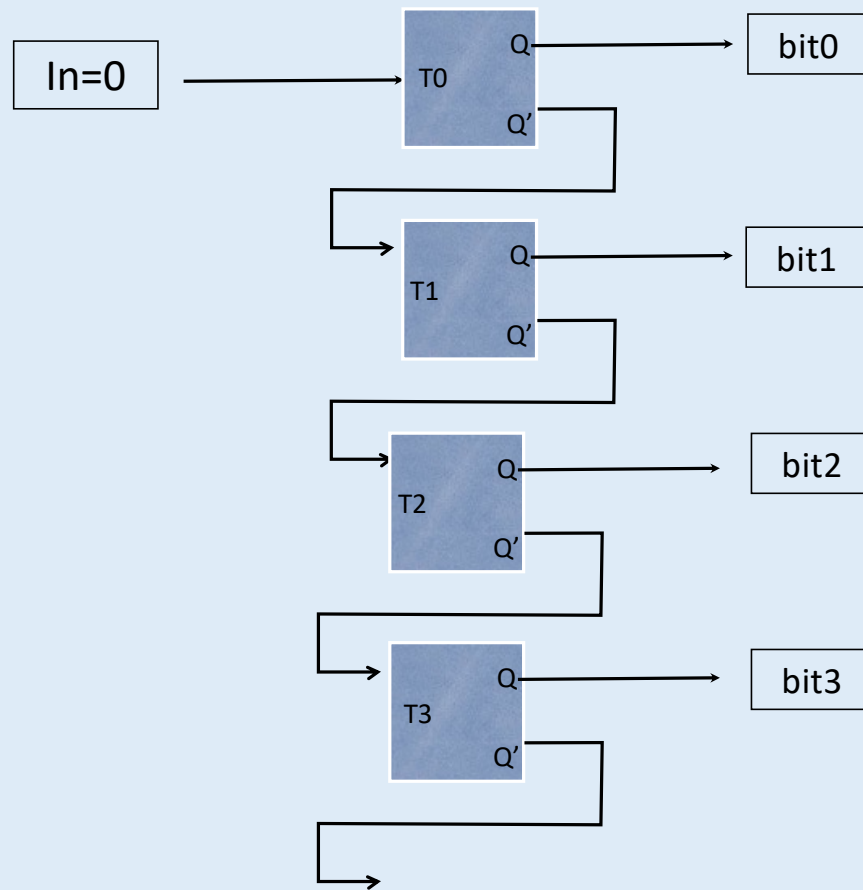
- At first rising edge of In
- T0 flips
 - $Q0$ goes to 1
 - $Q0'$ goes to 0
- What happens to T1?

Counting with Flip Flops



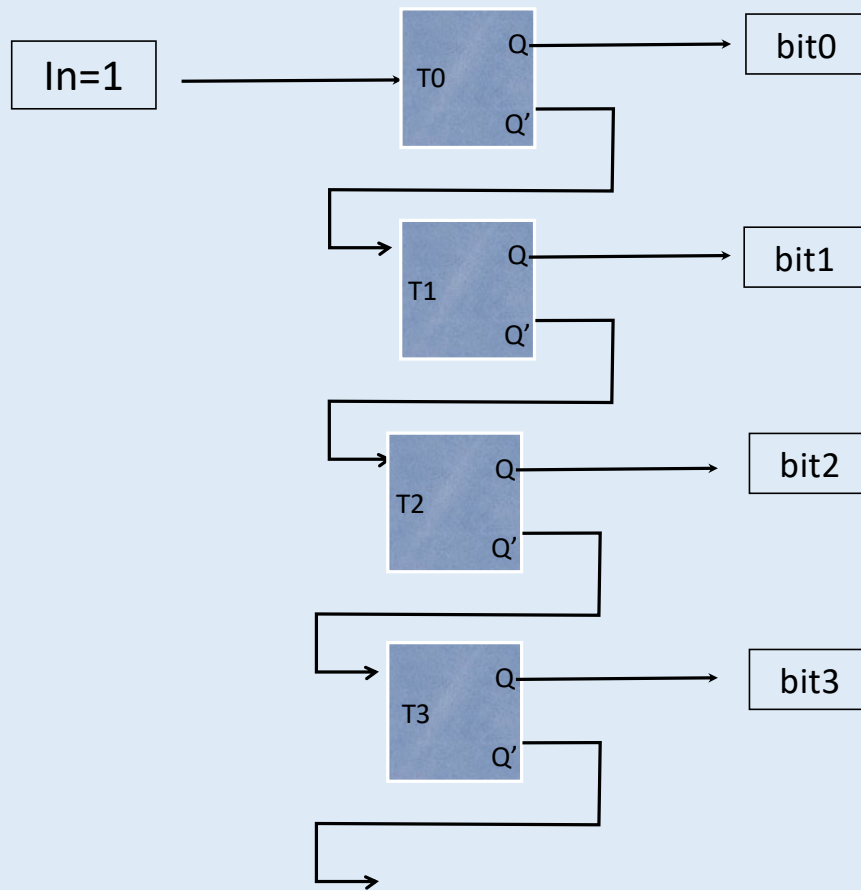
- At first rising edge of In
- T0 flips
 - Q0 goes to 1
 - Q0' goes to 0
- What happens to T1?
- **Nothing** happens to T1
- Output Q' of T0 did not change
- So we have
 - *bit0* = 1
 - *bit1* = 0
 - *bit2* = 0
 - *bit3* = 0

Counting with Flip Flops



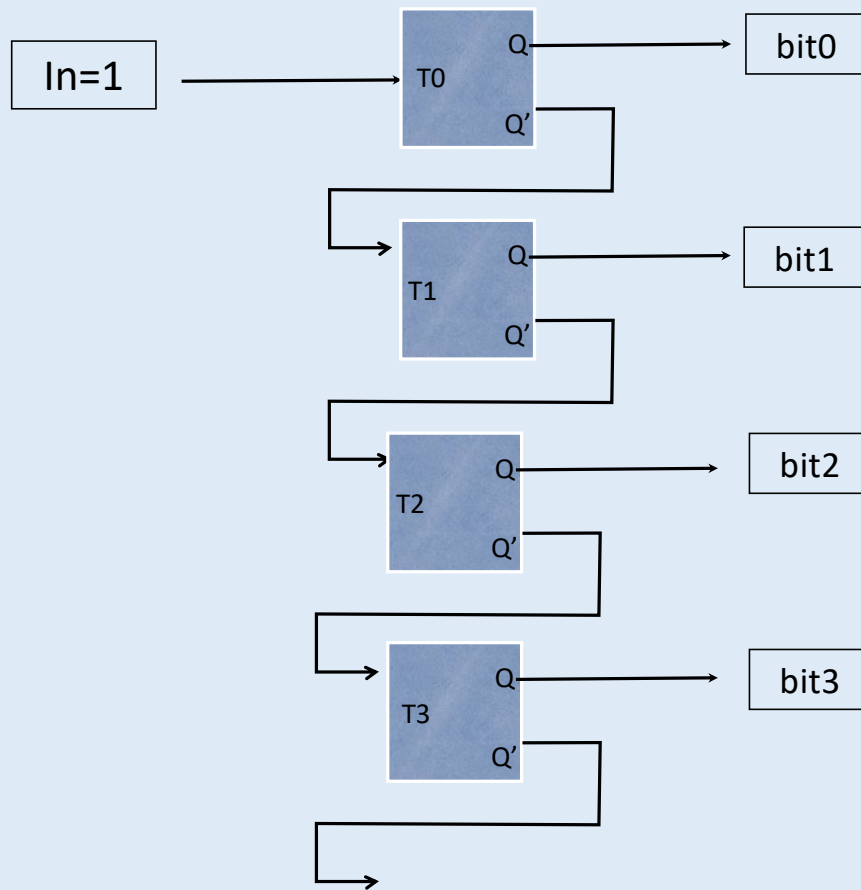
- No change when In goes to 0

Counting with Flip Flops



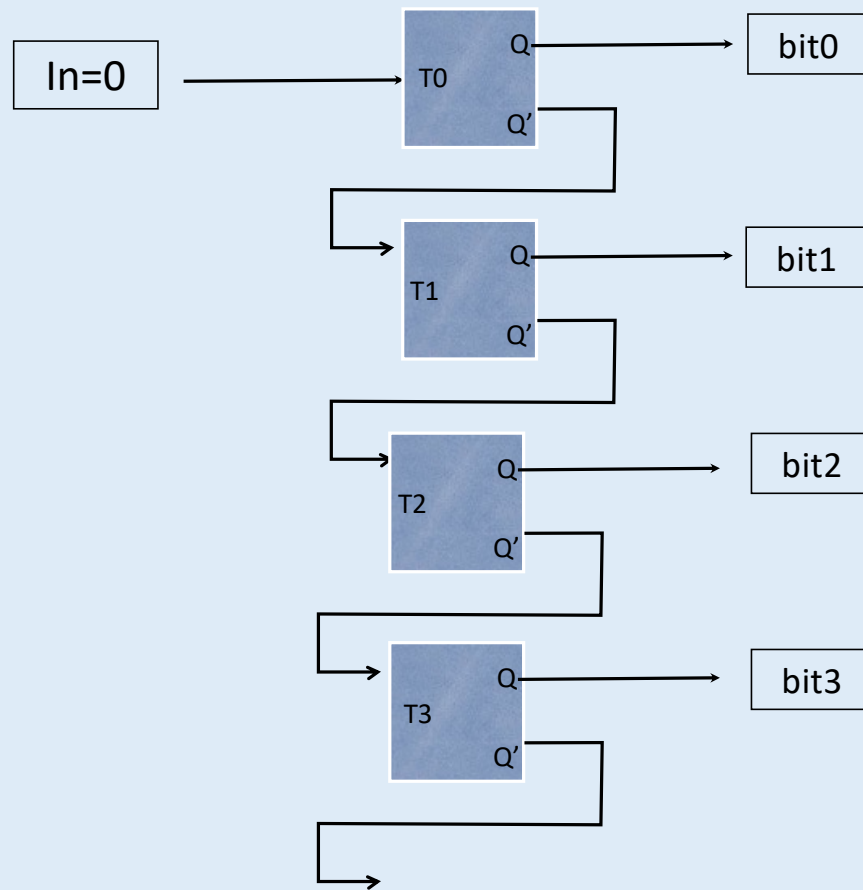
- At second rising edge of In
- T0 flips
 - Q0 goes to 0
 - Q0' goes to 1
- What happens to T1?

Counting with Flip Flops



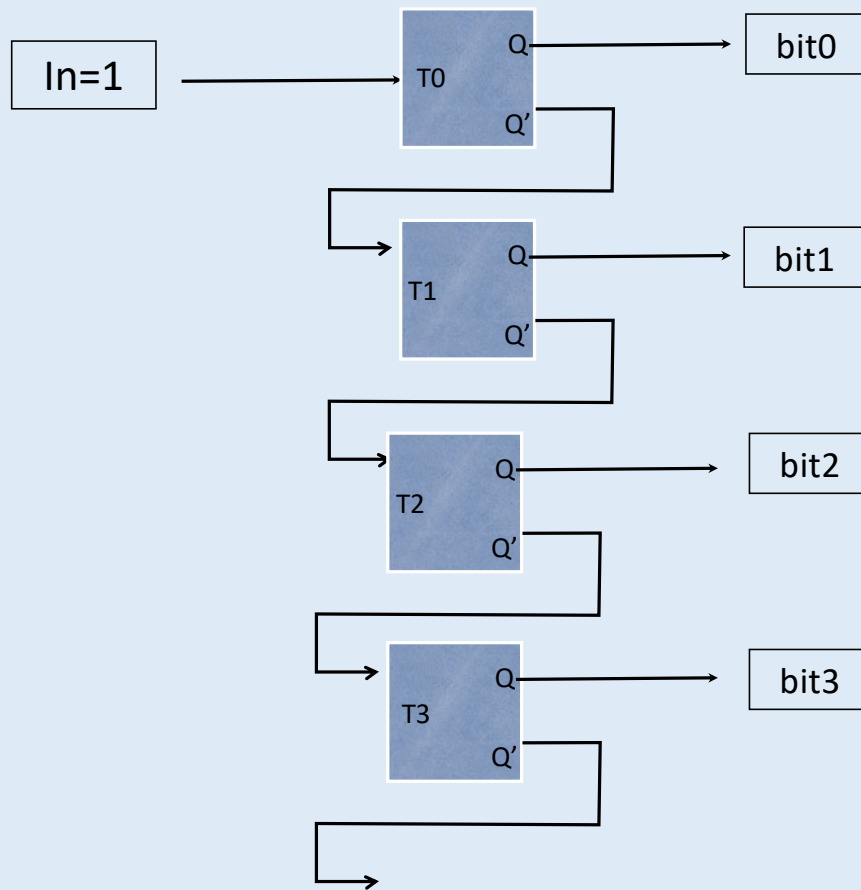
- At second rising edge of In
- T0 flips
 - Q0 goes to 0
 - Q0' goes to 1
- What happens to T1?
- **T1 flips**
- Because Q' of T0 changed
- So we have
 - $bit0 = 0$
 - $bit1 = 1$
 - $bit2 = 0$
 - $bit3 = 0$

Counting with Flip Flops



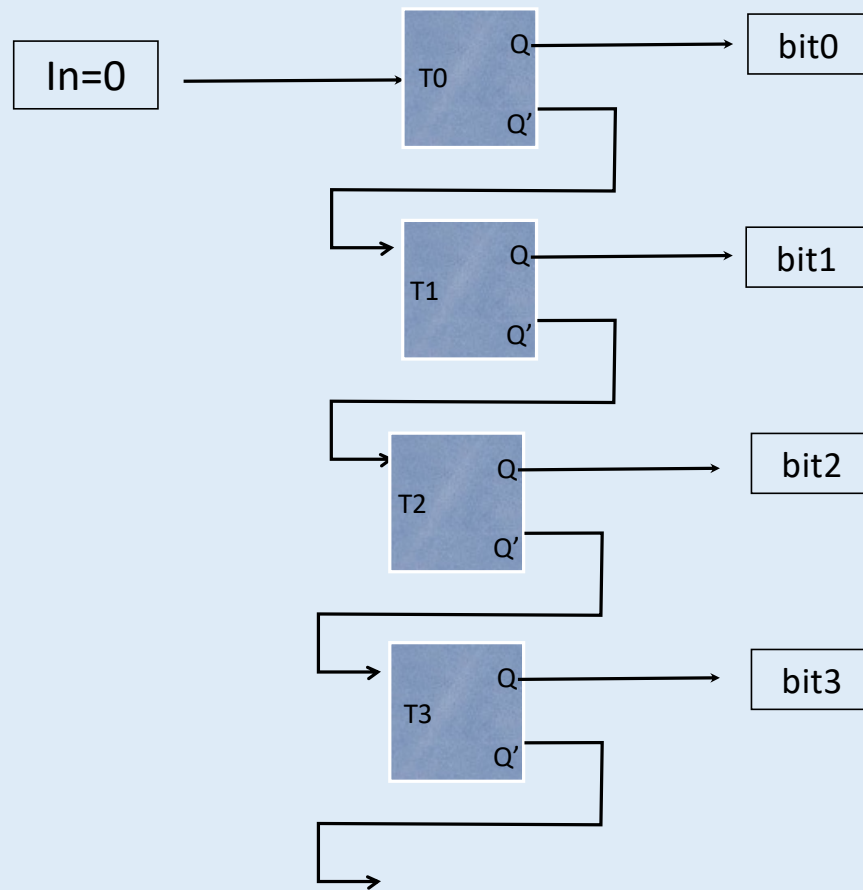
- No change when In goes to 0

Counting with Flip Flops



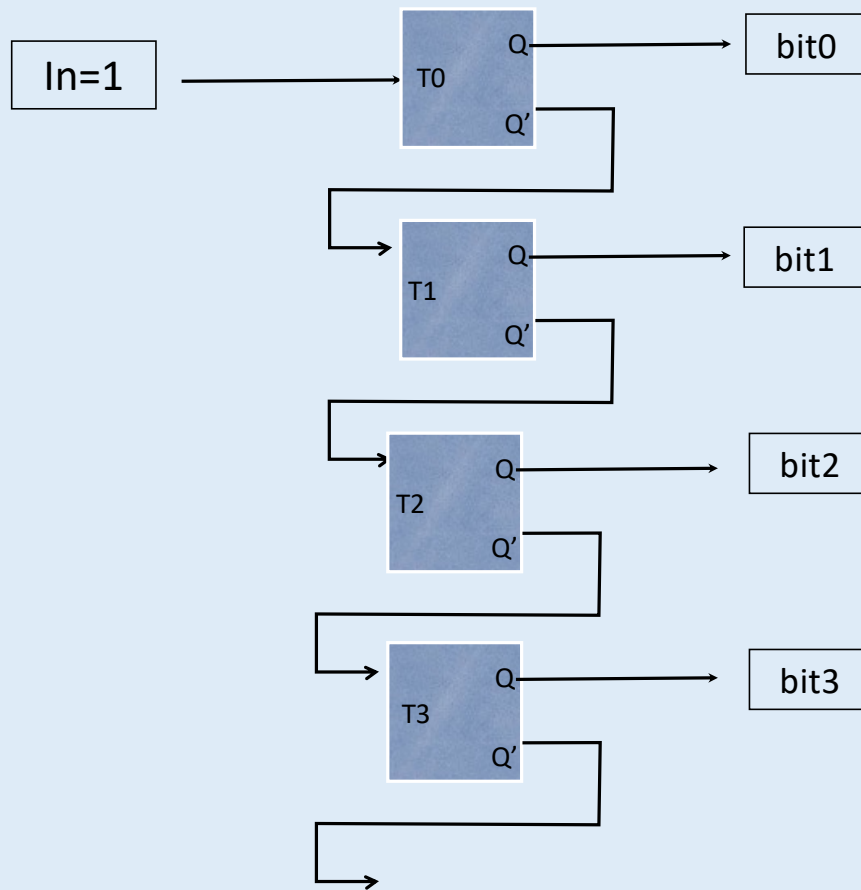
- At third rising edge of In
- T0 flips
 - $Q0$ goes to 1
 - $Q0'$ goes to 0
- T1 does *not* flip
- So we have
 - $bit0 = 1$
 - $bit1 = 1$
 - $bit2 = 0$
 - $bit3 = 0$

Counting with Flip Flops



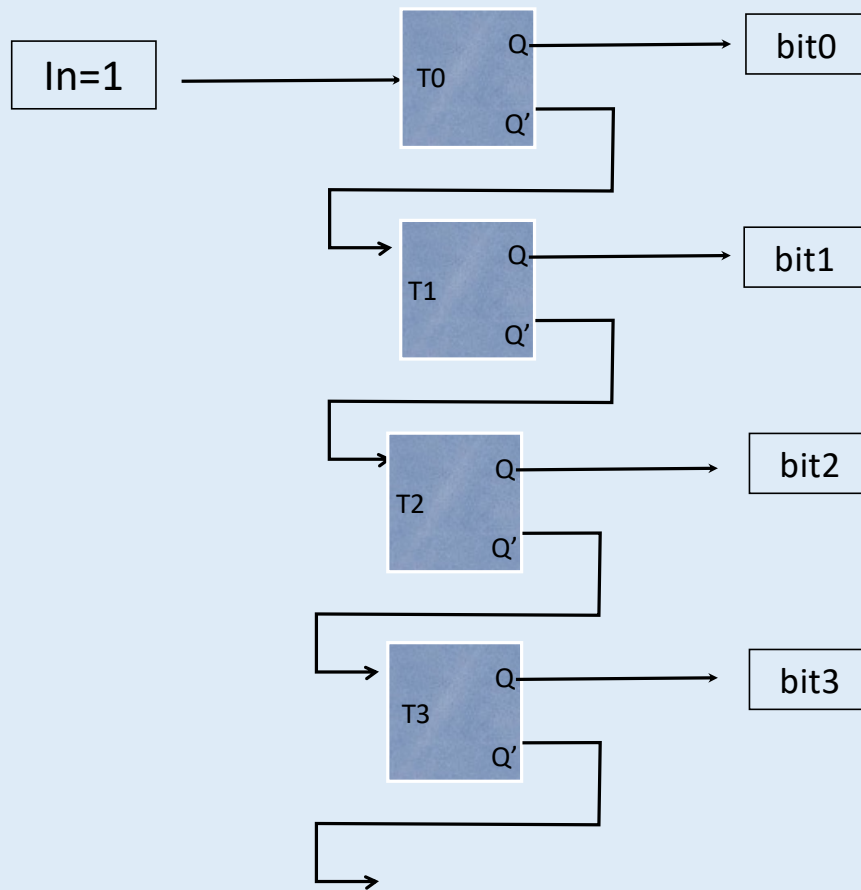
- No change when In goes to 0

Counting with Flip Flops



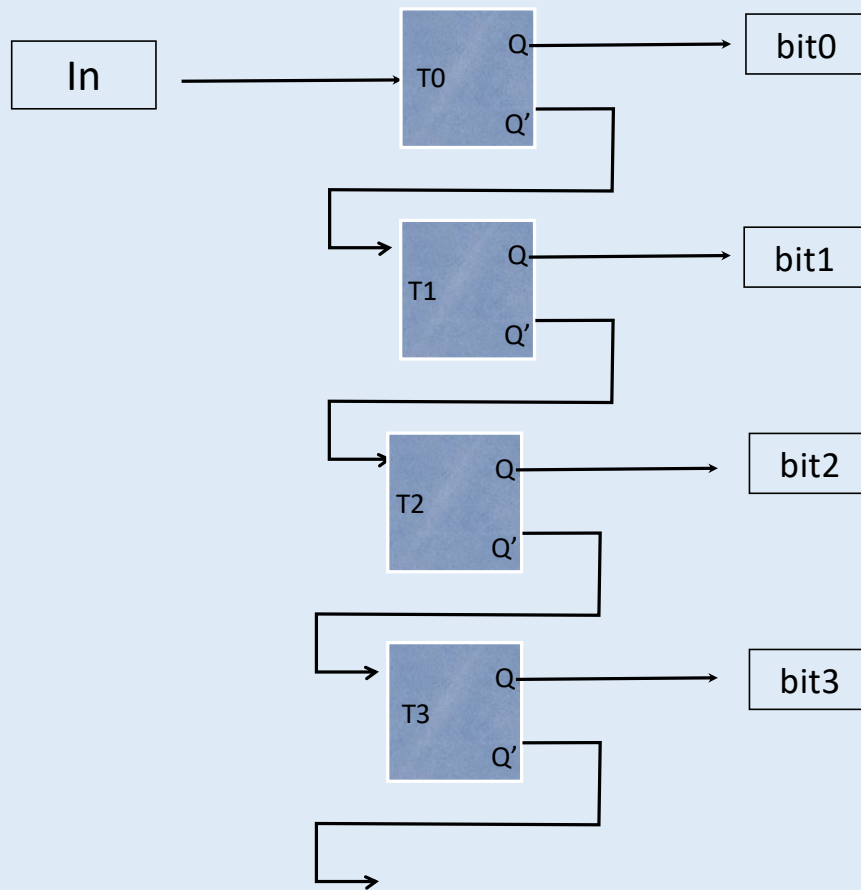
- At fourth rising edge of In
- T0 flips
 - Q0 goes to 0
 - Q0' goes to 1
- T1 now flip
 - Q1 goes to 0
 - Q1' goes to 1
- T2 now flip
 - Q2 goes to 1
 - Q2' goes to 0
- So we have
 - $bit0 = 0$
 - $bit1 = 0$
 - $bit2 = 1$
 - $bit3 = 0$

Counting with Flip Flops



- At fourth rising edge of In
- T0 flips
 - Q0 goes to 0
 - Q0' goes to 1
- T1 now flip
 - Q1 goes to 0
 - Q1' goes to 1
- T2 now flip
 - Q2 goes to 1
 - Q2' goes to 0
- So we have
 - $bit0 = 0$
 - $bit1 = 0$
 - $bit2 = 1$
 - $bit3 = 0$

Counting with Flip Flops



- This counts the number of rising edges at *In*
- i.e. the number of times *In* changes from 0 to 1
- As 4 bit number
 - *bit0* least significant
- Each flip flop halves the clock cycle
- If *In* is a clock then
 - *bit0* slows the clock by x2
 - *bit1* slows the clock by x4
 - *bit2* slows the clock by x8
 - *bit3* slows the clock by x16

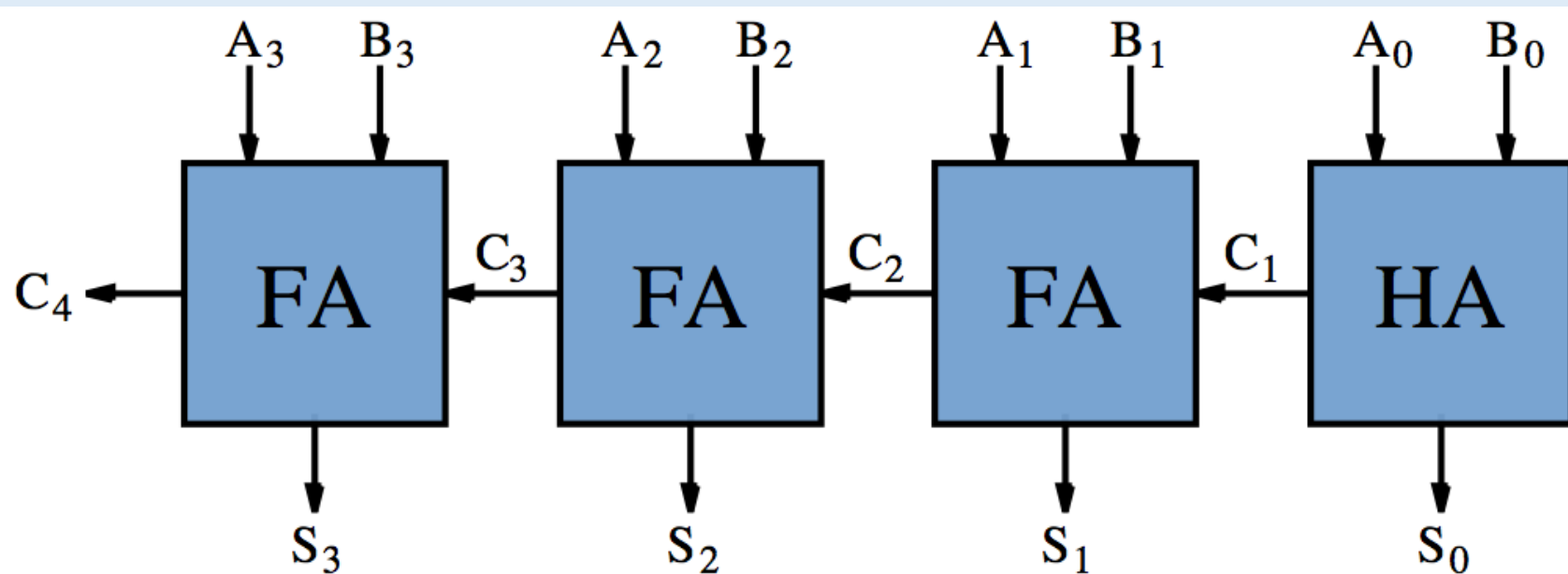
More Combinational Circuits

Carry Select Adder

Barrel Shifter

Carry Select Adder

Reminder: 4 Bit Ripple Adder



17 gates, 10 Gate delays,
157 and 94 for 32 bits,
317 and 190 for 64

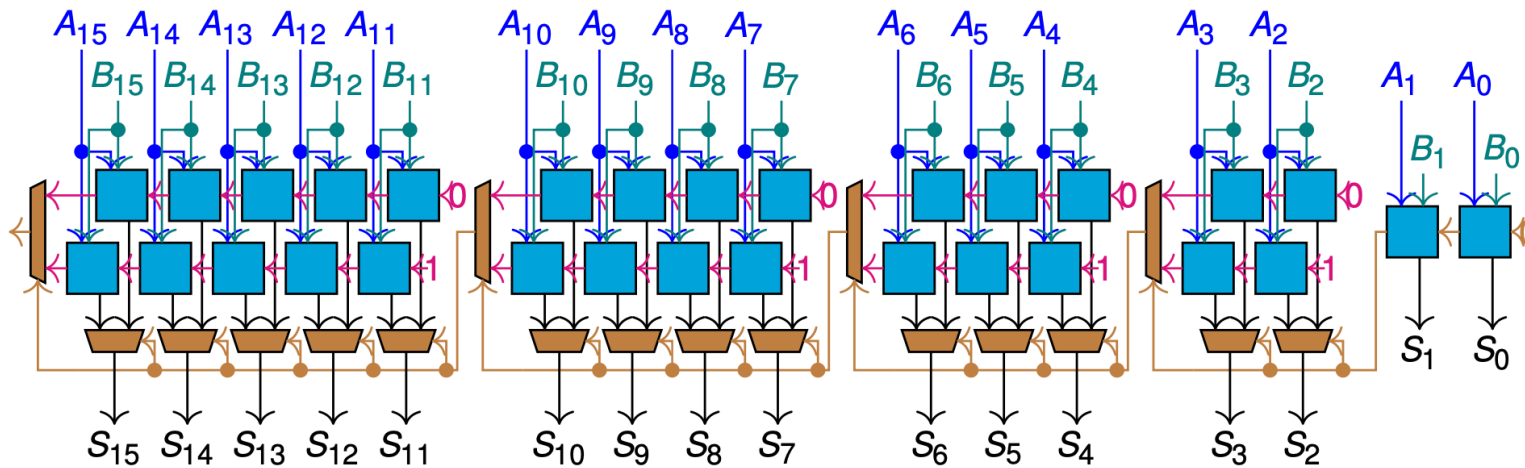
Carry-Select Adder

- The slowness of Ripple Carry is having signals go through all adders
 - E.g. 63 full adders and one half adder in a 64 bit adder
- Requires few gates but makes it slow
 - By using more gates, can we do it faster?
- Note the **tradeoff**
 - We might add more cost (gates and power to run them)
 - But improve performance (reduce time to stabilize)
 - Understanding tradeoffs is a key part of all areas of CS

Towards a faster adder

- Say We want a 16 bit adder
- Most significant (upper) 8 bits of the result determined by:
 - Upper 8 bits of each input
 - The carry bit from the addition of the least significant (lower) 8 bits
 - which is either 0 or 1
 - but we won't know which until the lower sum is finished
- So try computing two versions of the lower sum
 - one if carry is 0
 - another if carry is 1
 - once we know what the carry is, choose one and discard the other

16 Bit Carry-Select Adder



is full adder

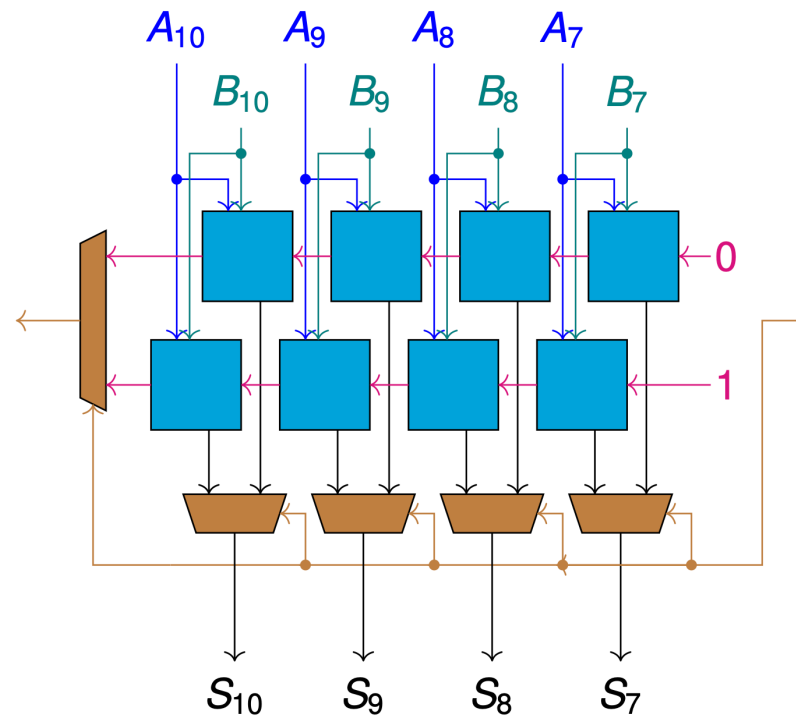


is 2-to-1 multiplexer


Perhaps not totally self explanatory!

Fragment in Detail

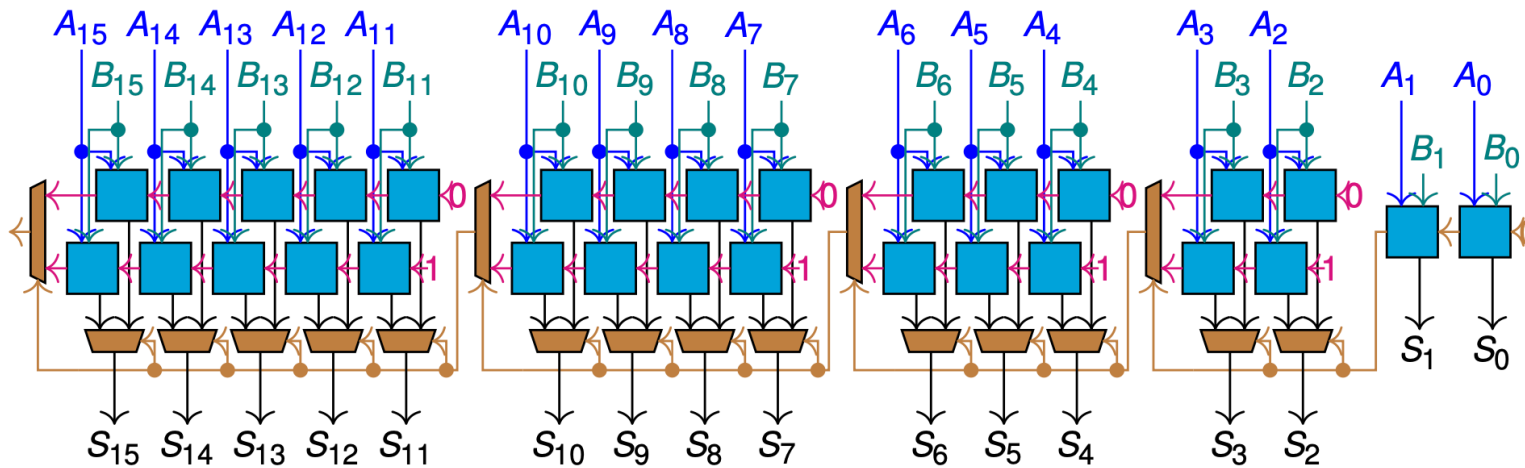
- Two 4-bit ripple adders
- Running in parallel
- One with 0 carry in
- One with 1 carry in
- Wire in on right tells us the carry from less significant bits
- 4 Multiplexers use that to choose the adder to take output of
- 5th Multiplexer selects carry bit to go to next part of circuit



 is full adder

 is 2-to-1 multiplexer

16 Bit Carry-Select Adder



is full adder



is 2-to-1 multiplexer

Note this is not a simple $8+8=16$ split

It's split as $5 + 4 + 3 + 2 + 2=16$

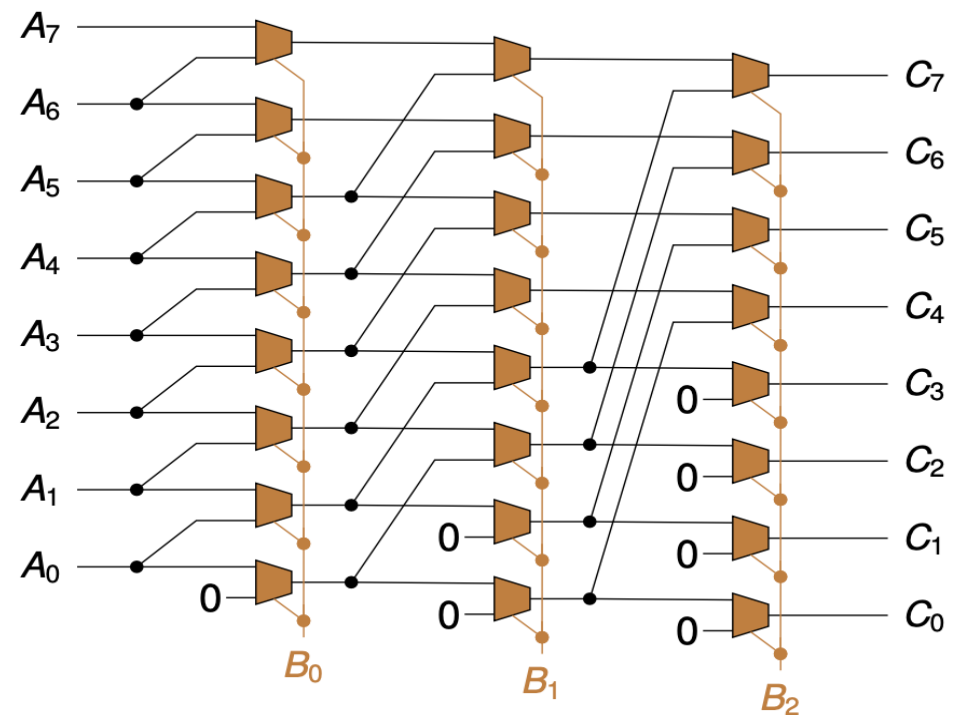
Barrel Shifter

Barrel Shifter

- Common operation is left or right shift
 - Corresponds to multiplication or division by power of 2
 - Left shift is << in C and right shift is >>
- We wish to shift all bits simultaneously
 - And with circuit depth independent of the shift amount
 - E.g. in 8 bit barrel shifter we want shift of 1 or 7 to take the same time
- This will be much faster than general multiplication and division
 - Also used in implementation of floating-point arithmetic

Barrel Shifter

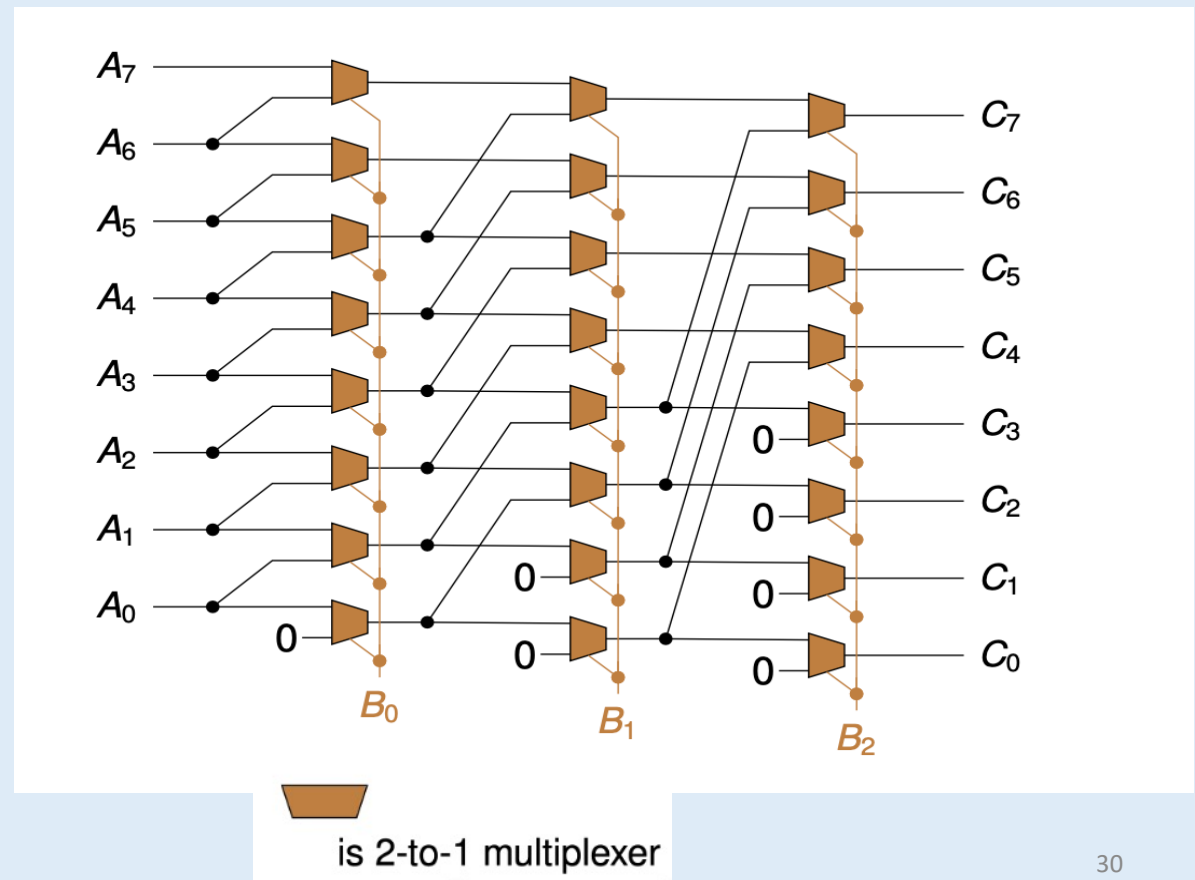
- Example implements $c = a \ll b$
 - c and a are 8 bits
 - Only 3 bits of b matter
 - Since we can't shift more than 7
- The bits shifted in are all 0



is 2-to-1 multiplexer

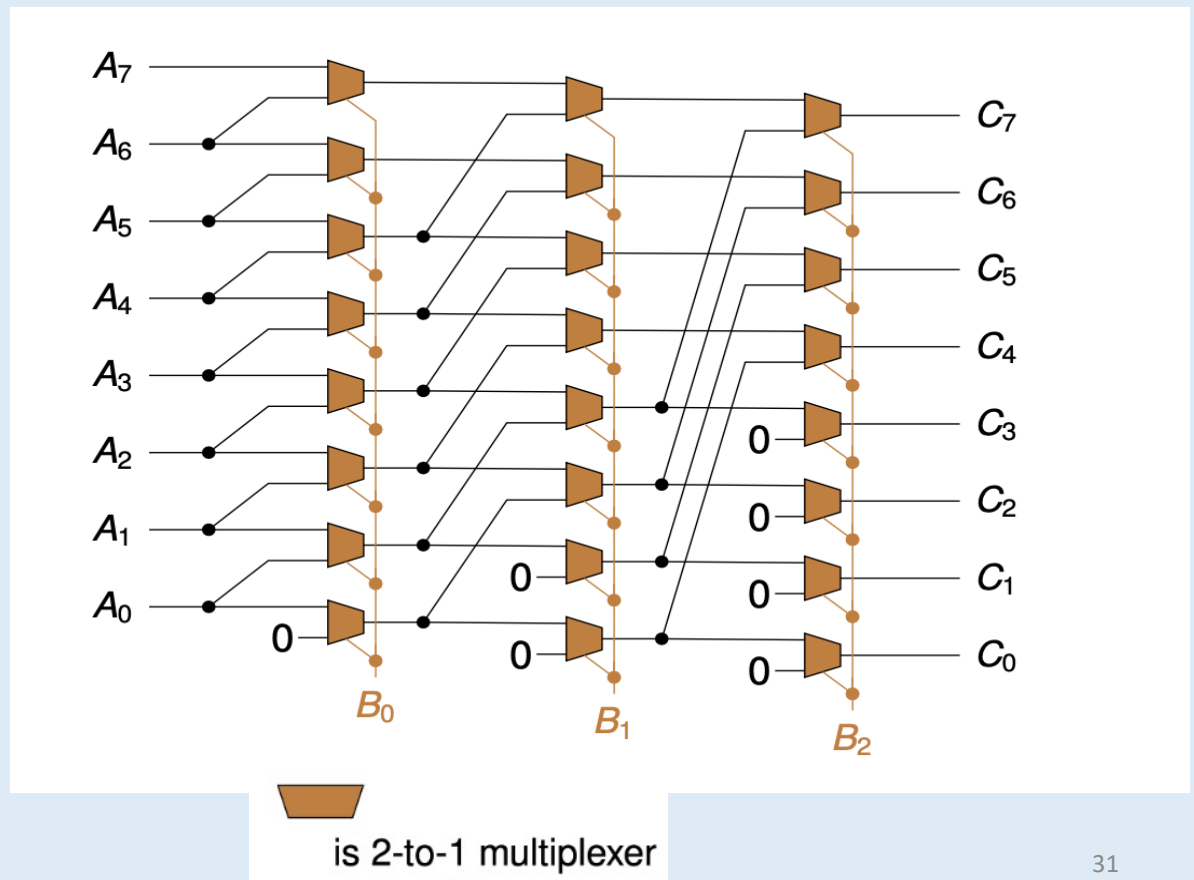
Barrel Shifter

- Look at least significant bit B_0
- Column of multiplexers
- $B_0 = 0$: leave A bit in same row
- $B_0 = 1$: move A bit one row up



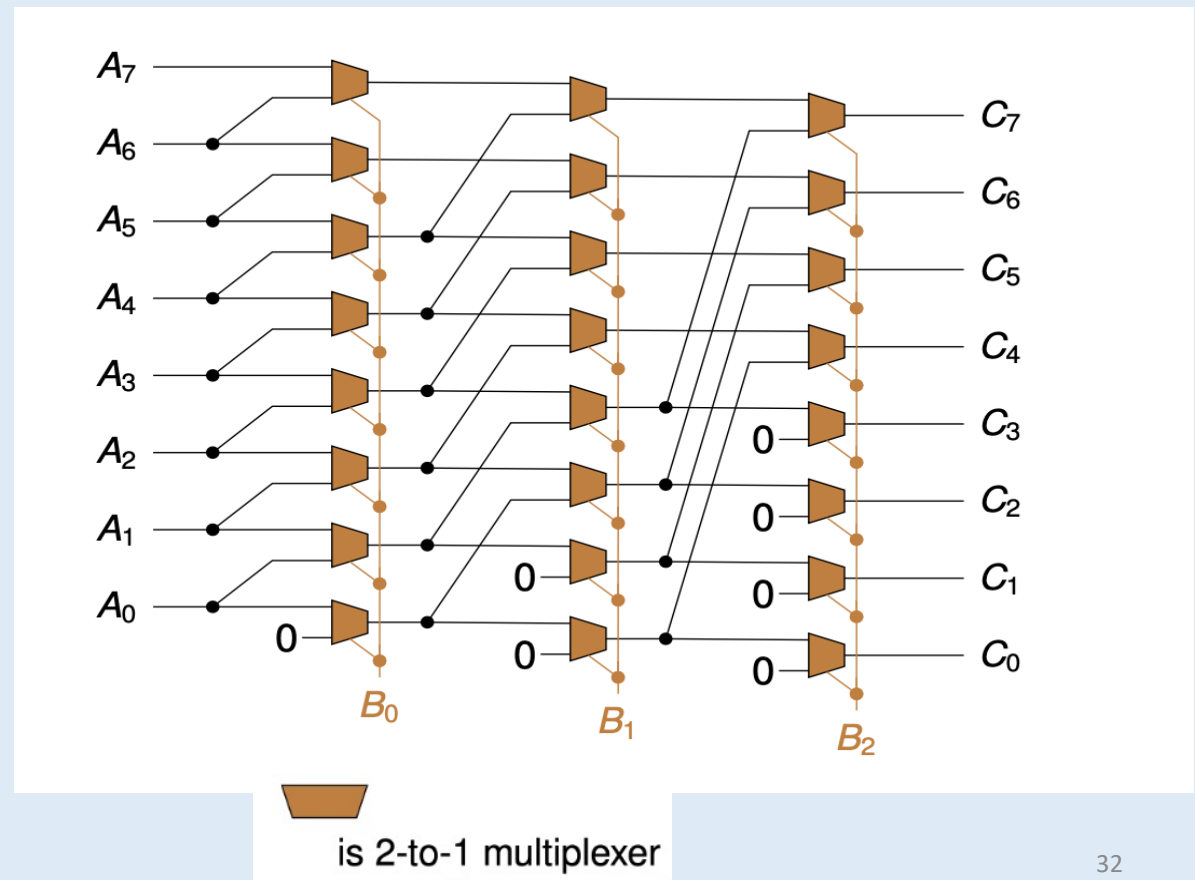
Barrel Shifter

- Look at least significant bit B_0
- Column of multiplexers
- $B_0 = 0$: leave A bit in same row
- $B_0 = 1$: move A bit **one row up**



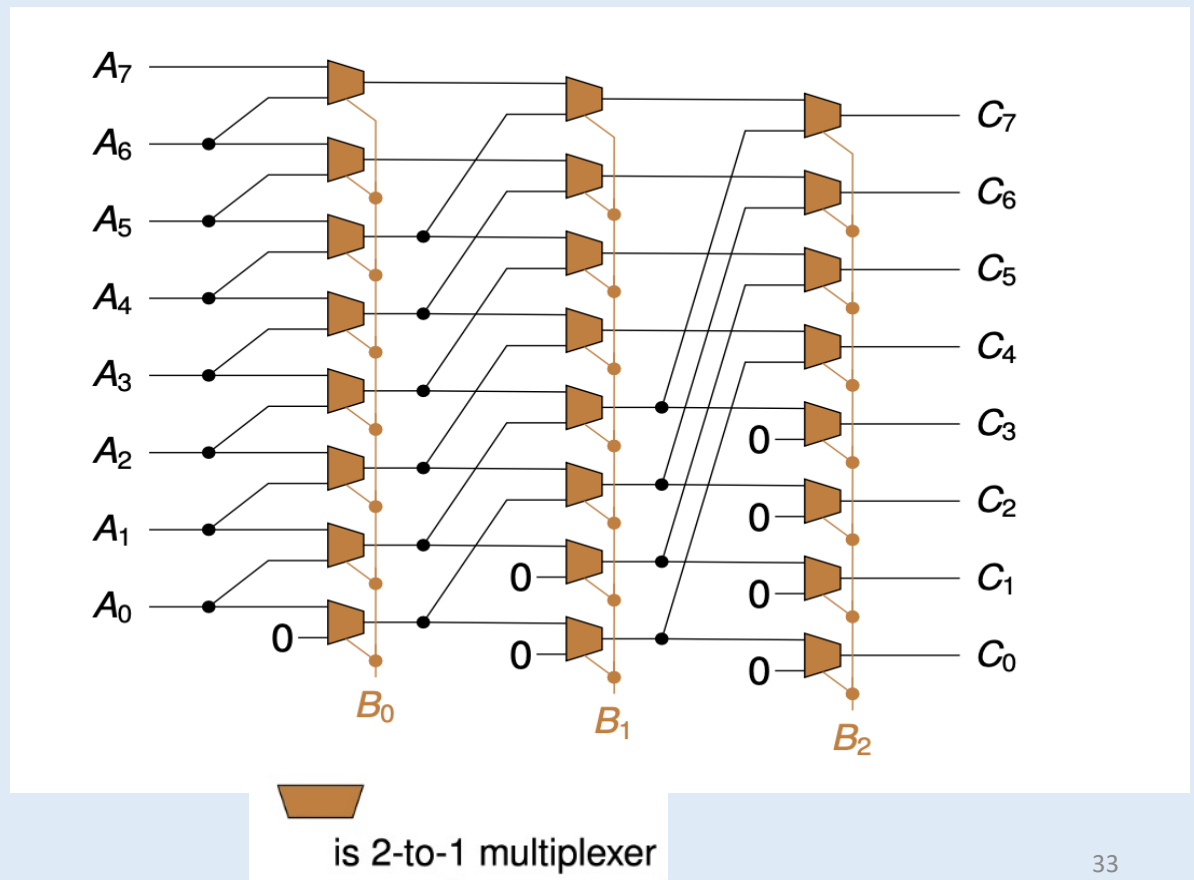
Barrel Shifter

- Look at next bit B_1
- Column of multiplexers
- $B_1 = 0$: leave A bit in same row
- $B_1 = 1$: move A bit **two rows up**



Barrel Shifter

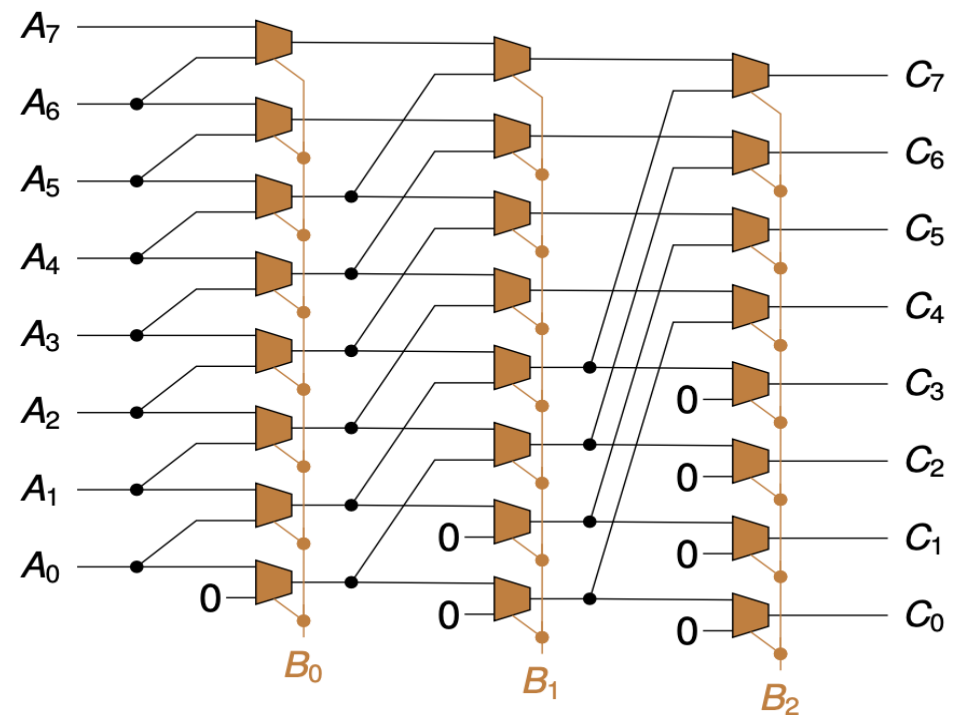
- Look at highest bit B_2
- Column of multiplexers
- $B_2 = 0$: leave A bit in same row
- $B_2 = 1$: move A bit **four rows up**



Barrel Shifter

End result is that each A bit will move up the number of bits represented by B

With 0 replacing any high bits

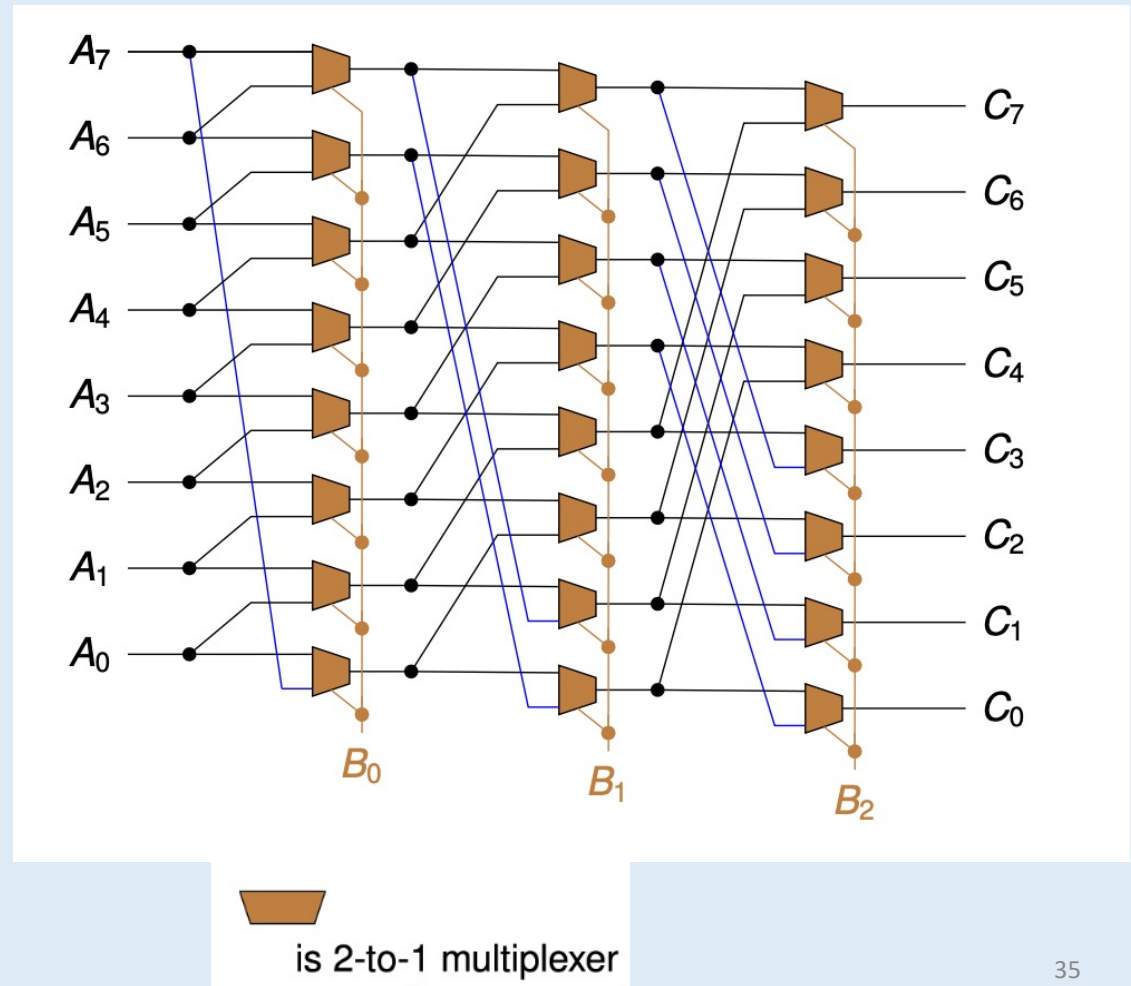


is 2-to-1 multiplexer

Barrel *Cyclic* Shifter

Like previous shifter

But high bits replaced by original low bits



That's all for Logic for now

- Lots and lots more to enjoy and love in later modules in your degree!