EECS150 - Digital Design
Lecture 25 – Shifters and Counters

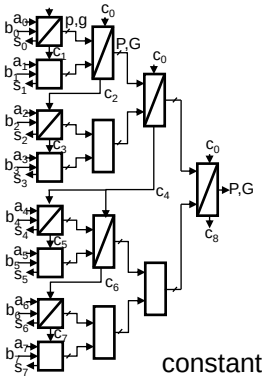
Nov. 21, 2013
Prof. Ronald Fearing
Electrical Engineering and Computer Sciences
University of California, Berkeley

(slides courtesy of Prof. John Wawrzynek)

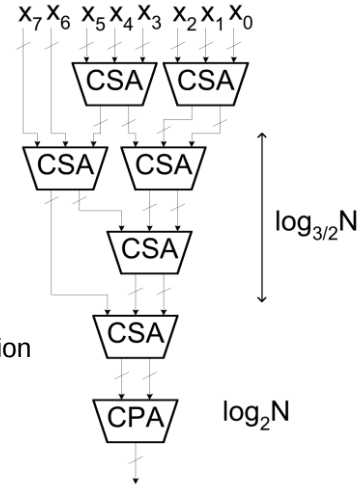
http://www-inst.eecs.berkeley.edu/~cs150

Recap

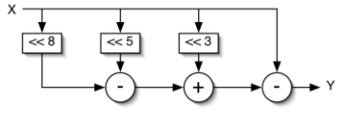
- Carry Look-ahead Adder



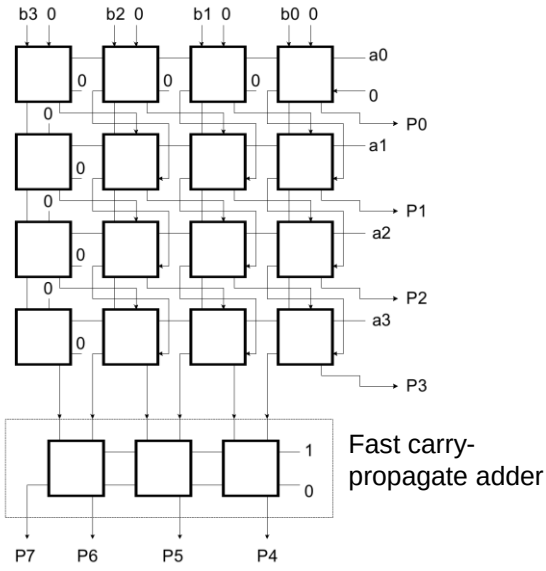
Carry save adder



constant coefficient multiplication

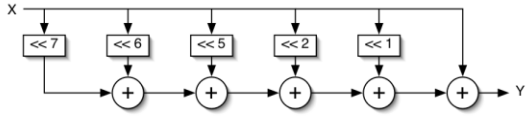


Recap: multiply using carry save addition

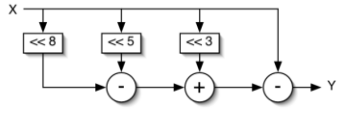


Recap "Constant Coefficient Multiplication" (KCM)

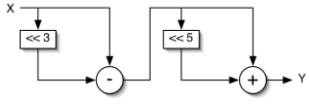
Binary multiplier: $Y = 231*X = (2^7 + 2^6 + 2^5 + 2^2 + 2^1+2^0)*X$



- CSD helps, but the multipliers are limited to shifts followed by adds.
  - CSD multiplier: $Y = 231*X = (2^8 - 2^5 + 2^3 - 2^0)*X$



- How about shift/add/shift/add …?
  - KCM multiplier: $Y = 231*X = 7*33*X = (2^3 - 2^0)*(2^5 + 2^0)*X$

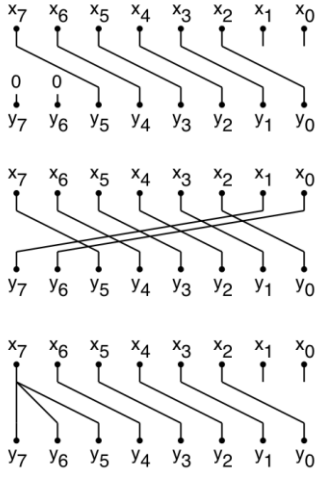


- No simple algorithm exists to determine the optimal KCM representation.
- Most use exhaustive search method.

Outline

- *Shifters*
- *Counters*

Fixed Shifters / Rotators



Logical Shift

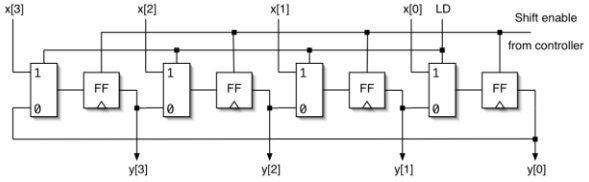
Rotate

Arithmetic Shift

Variable Shifters / Rotators

- Example: X >> S, where S is unknown when we synthesize the circuit.
- Uses: shift instruction in processors (ARM includes a shift on every instruction), floating-point arithmetic, division/multiplication by powers of 2, etc.
- One way to build this is a simple shift-register:

  a) Load word, b) shift enable for S cycles, c) read word.

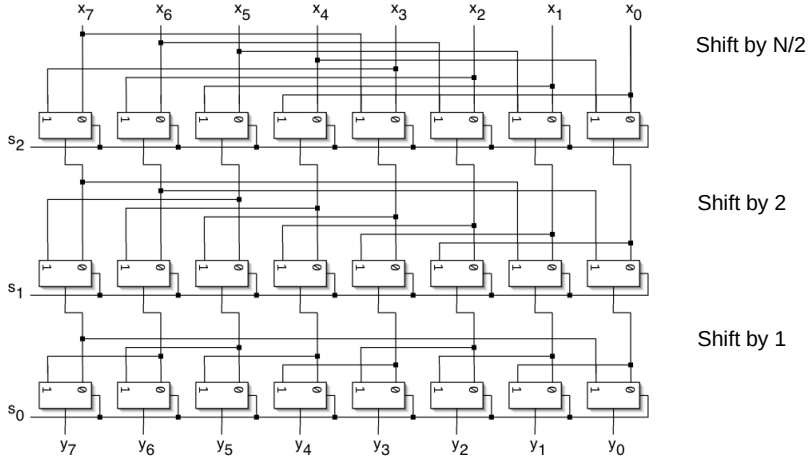


– Worst case delay O(N) , not good for processor design.

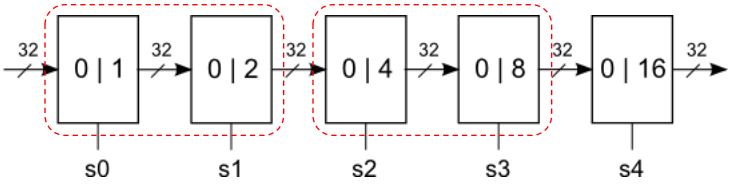
– Can we do it in O(logN) time and fit it in one cycle?

Log Shifter / Rotator

- Log(N) stages, each shifts (or not) by a power of 2 places, S=[s_2;s_1;s_0]:

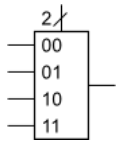


LUT Mapping of Log shifter

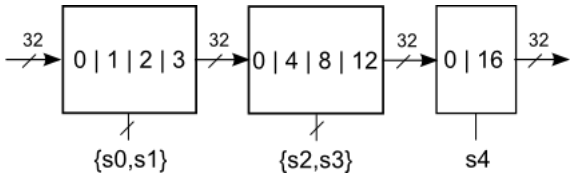


Efficient with 2to1 multiplexors, for instance, 3LUTs.

Virtex5 has 6LUTs. Naturally makes 4to1 muxes:



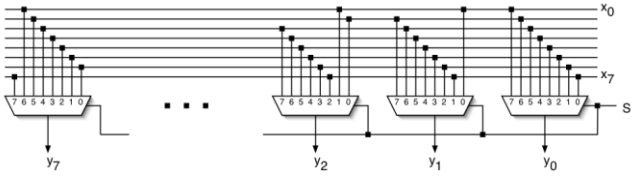
Reorganize shifter to use 4to1 muxes.



Final stage uses F7 mux

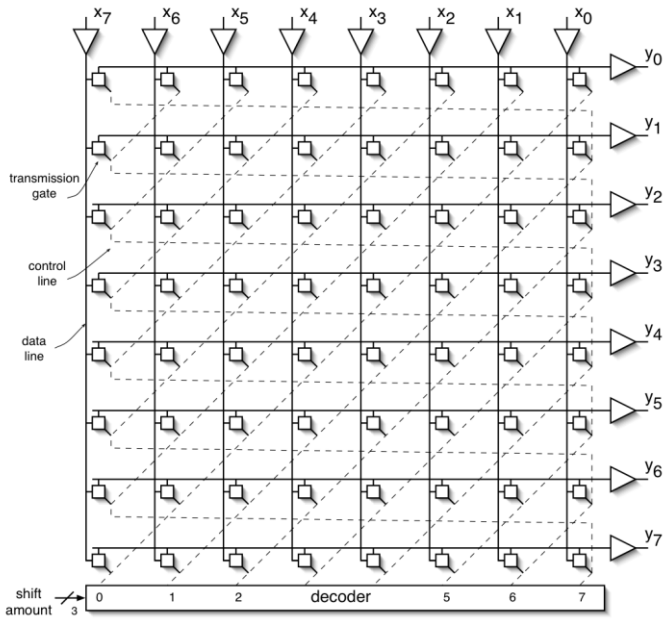
"Improved" Shifter / Rotator

- How about this approach? Could it lead to even less delay?



- What is the delay of these big muxes?
- Look at transistor-level implementation?

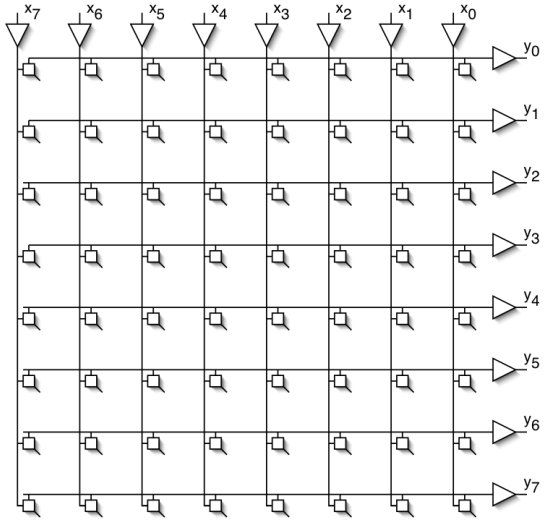
Barrel Shifter



Cost/delay?

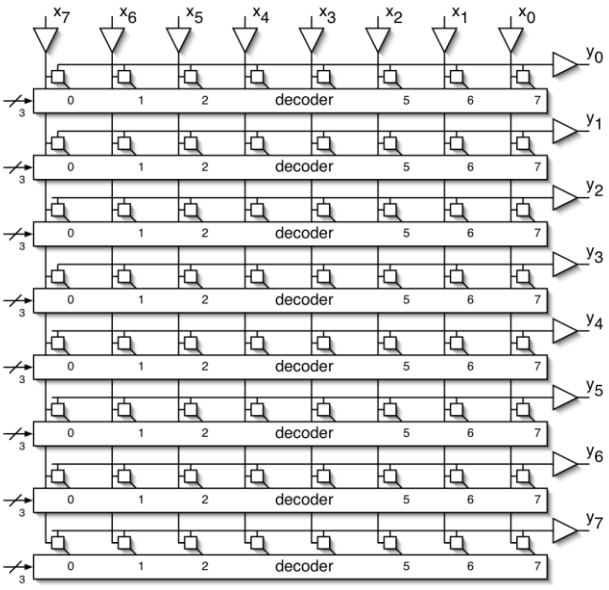
- (don't forget the decoder)

Connection Matrix



- Generally useful structure:
  - N^2 control points.
  - What other interesting functions can it do?

Cross-bar Switch



Nlog(N) control signals.

Supports all interesting permutations

- All one-to-one and one-to-many connections.

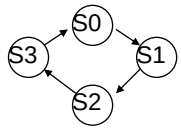
Commonly used in communication hardware (switches, routers).

Outline

- *Shifters*
- **Counters**

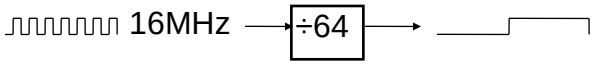
Counters

- Special sequential circuits (FSMs) that repeatedly sequence through a set of outputs.
- Examples:
  - binary counter: 000, 001, 010, 011, 100, 101, 110, 111, 000,
  - gray code counter:
    000, 010, 110, 100, 101, 111, 011, 001, 000, 010, 110, …
  - one-hot counter: 0001, 0010, 0100, 1000, 0001, 0010, …
  - BCD counter: 0000, 0001, 0010, …, 1001, 0000, 0001
  - pseudo-random sequence generators: 10, 01, 00, 11, 10, 01, 00, ...
- Moore machines with “ring” structure in State Transition Diagram:



What are they used for?

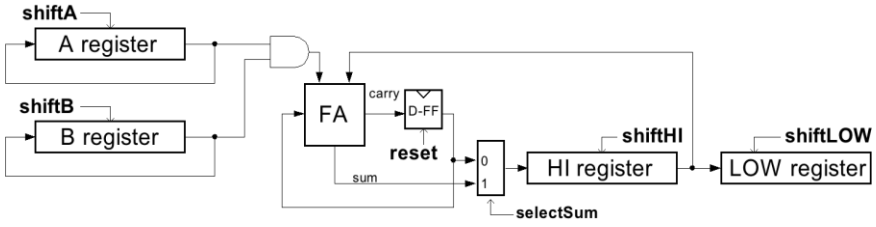
- Counters are commonly used in hardware designs because most (if not all) computations that we put into hardware include iteration (looping). Examples:
  - Shift-and-add multiplication scheme.
  - Bit serial communication circuits (must count one “words worth” of serial bits.
- Other uses for counter:
  - Clock divider circuits



  - Systematic inspection of data-structures
    - Example: Network packet parser/filter control.
- Counters simplify “controller” design by:
  - providing a specific number of cycles of action,
  - sometimes used with a decoder to generate a sequence of timed control signals.
  - Consider using a counter when many FSM states with few branches.

Controller using Counters

- Example, Bit-serial multiplier (n^2 cycles, one bit of result per n cycles):



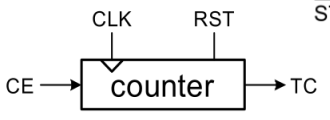
- Control Algorithm:

```
repeat n cycles {   // outer (i) loop
        repeat n cycles{    // inner (j) loop
                shiftA, selectSum, shiftHI
        }
        shiftB, shiftHI, shiftLOW, reset
}
```

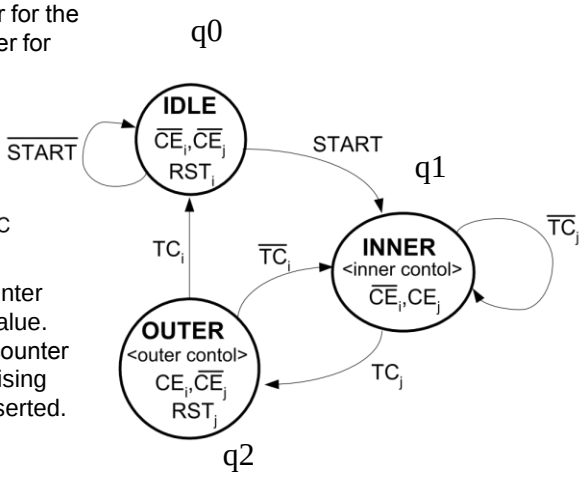
Note: The occurrence of a control signal x means x=1. The absence of x means x=0.

Controller using Counters

- **State Transition Diagram:**
  - Assume presence of two binary counters. An "i" counter for the outer loop and "j" counter for inner loop.

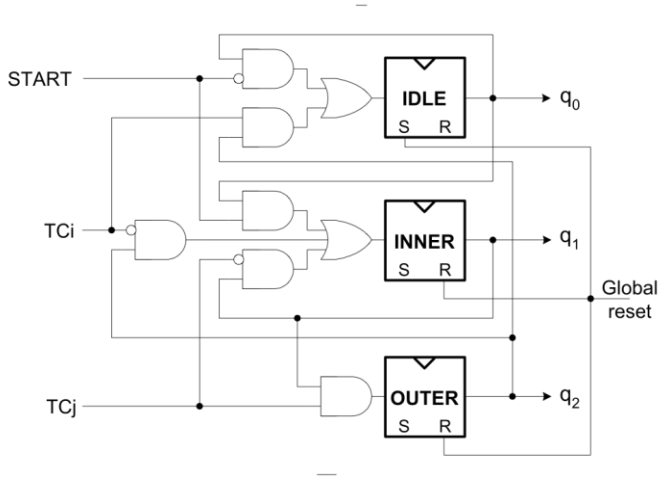


TC is asserted when the counter reaches it maximum count value. **CE** is "count enable". The counter increments its value on the rising edge of the clock if CE is asserted.



Controller using Counters

- **Controller circuit implementation:**



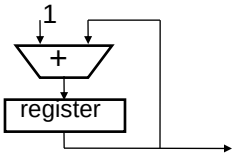
- Outputs:

$CE_i = q_2$
$CE_j = q_1$
$RST_i = q_0$
$RST_j = q_2$

$shiftA = q_1$
$shiftB = q_2$
$shiftLOW = q_2$
$shiftHI = q_1 + q_2$
$reset = q_2$
$selectSUM = q_1$

How do we design counters?

- For binary counters (most common case) incrementer circuit would work:



- In Verilog, a counter is specified as: x = x+1;
  - This does *not* imply an adder
  - An incrementer is simpler than an adder
  - And a counter might be simpler yet.

- In general, the best way to understand counter design is to think of them as FSMs, and follow general procedure, however some special cases can be optimized.

Synchronous Counters

All outputs change with clock edge.

- Binary Counter Design:

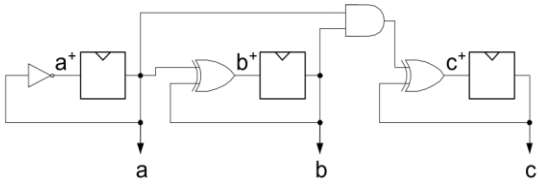
  Start with 3-bit version and generalize:

c	b	a	c^+	b^+	a^+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

$a^+ = a'$

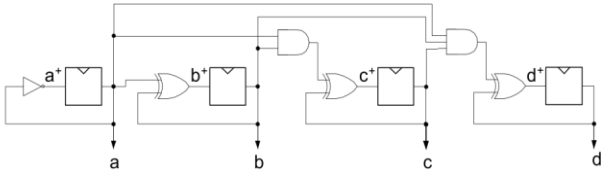
$b^+ = a \text{ xor } b$

$c^+ = abc' + a'b'c + ab'c + a'bc$
$= a'c + abc' + b'c$
$= c(a'+b') + c'(ab)$
$= c(ab)' + c'(ab)$
$= c \text{ xor } ab$

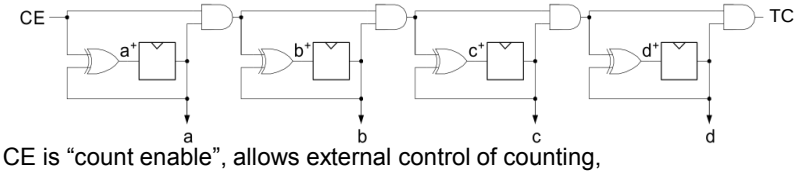


Synchronous Counters

- How do we extend to n-bits?
- Extrapolate c^+: $d^+ = d \text{ xor } abc$, $e^+ = e \text{ xor } abcd$

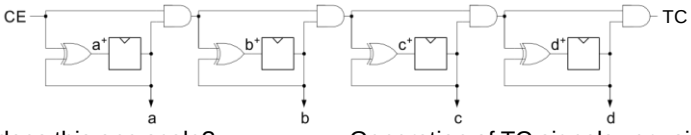


- Has difficulty scaling (AND gate inputs grow with n)



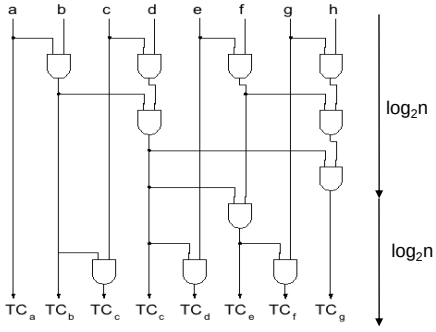
- CE is “count enable”, allows external control of counting,
- TC is “terminal count”, is asserted on highest value, allows cascading, external sensing of occurrence of max value.

Synchronous Counters

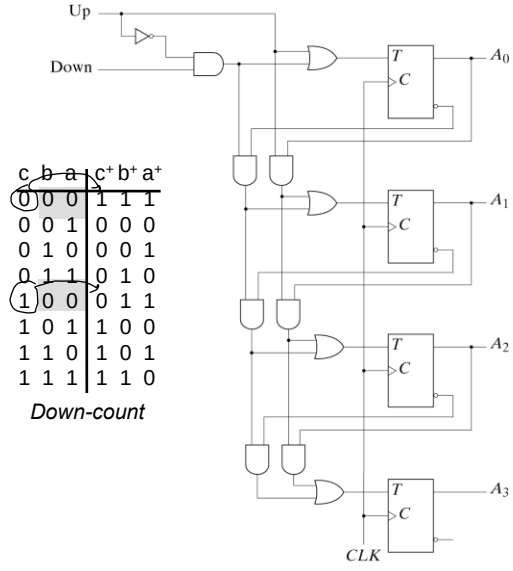


- How does this one scale?
- ☹ Delay grows ~ n

- Generation of TC signals very similar to generation of carry signals in adder.
- "Parallel Prefix" circuit reduces delay:

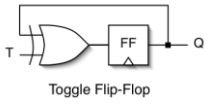


Up-Down Counter



c	b	a	c+	b+	a+
0	0	0	1	1	1
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Down-count

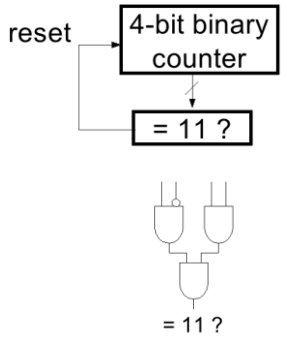


Toggle Flip-Flop

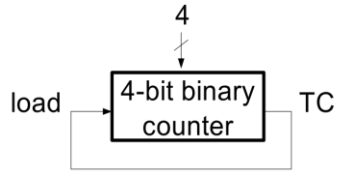
Fig. 6-13 4-Bit Up-Down Binary Counter

Odd Counts

- Extra combinational logic can be added to terminate count before max value is reached:
- Example: **count to 12**



- Alternative:

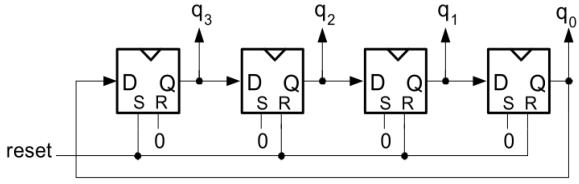


Ring Counters

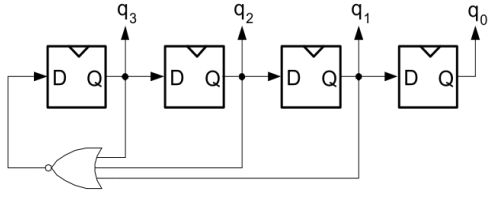
- "one-hot" counters

0001, 0010, 0100, 1000, 0001, …

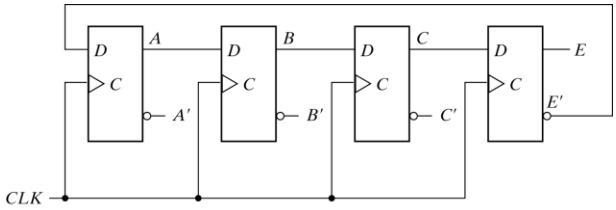
- What are these good for?



"Self-starting" version:



Johnson Counter



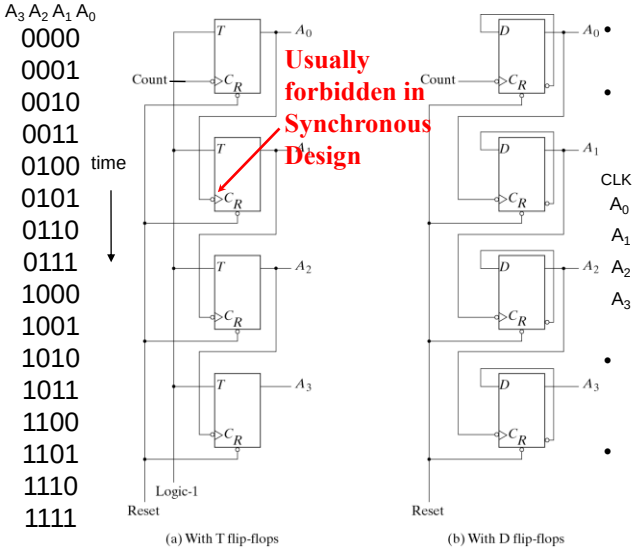
(a) Four-stage switch-tail ring counter

Sequence number	Flip-flop outputs A	B	C	E	AND gate required for output
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Count sequence and required decoding

Fig. 6-18 Construction of a Johnson Counter

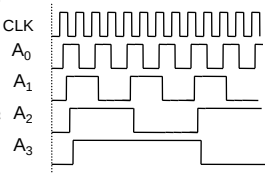
Asynchronous "Ripple" counters



(a) With T flip-flops (b) With D flip-flops

Fig. 6-8 4-Bit Binary Ripple Counter

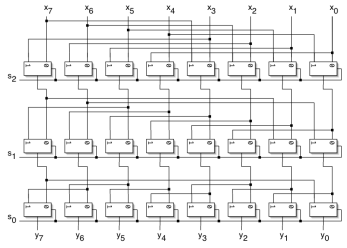
- Each stage is ÷2 of previous.
- Look at output waveforms:



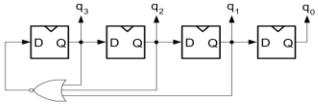
- Often called "asynchronous" counters.
- A "T" flip-flop is a "toggle" flip-flop. Flips state on cycles when T=1.

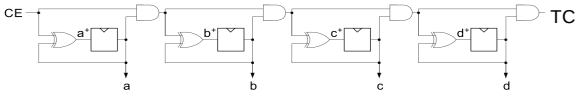
Summary

- Shifters



- Counters





29