

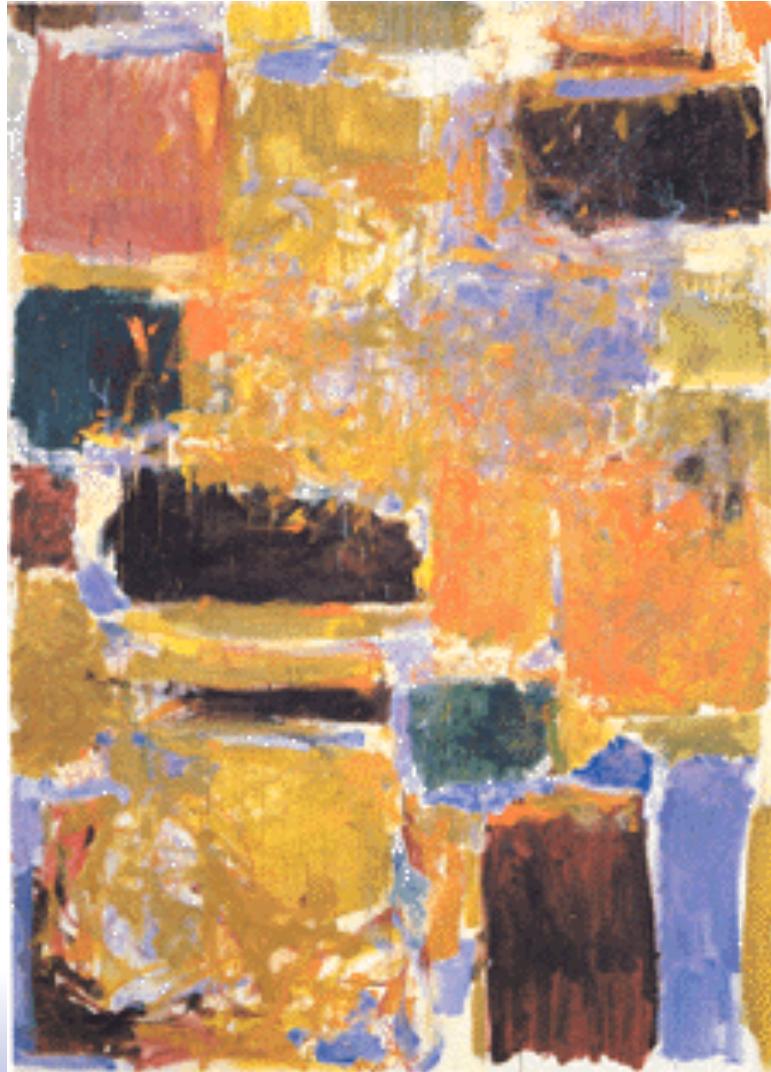


CS150 - EE141/241A
Fall 2014
Digital Design and
Integrated Circuits

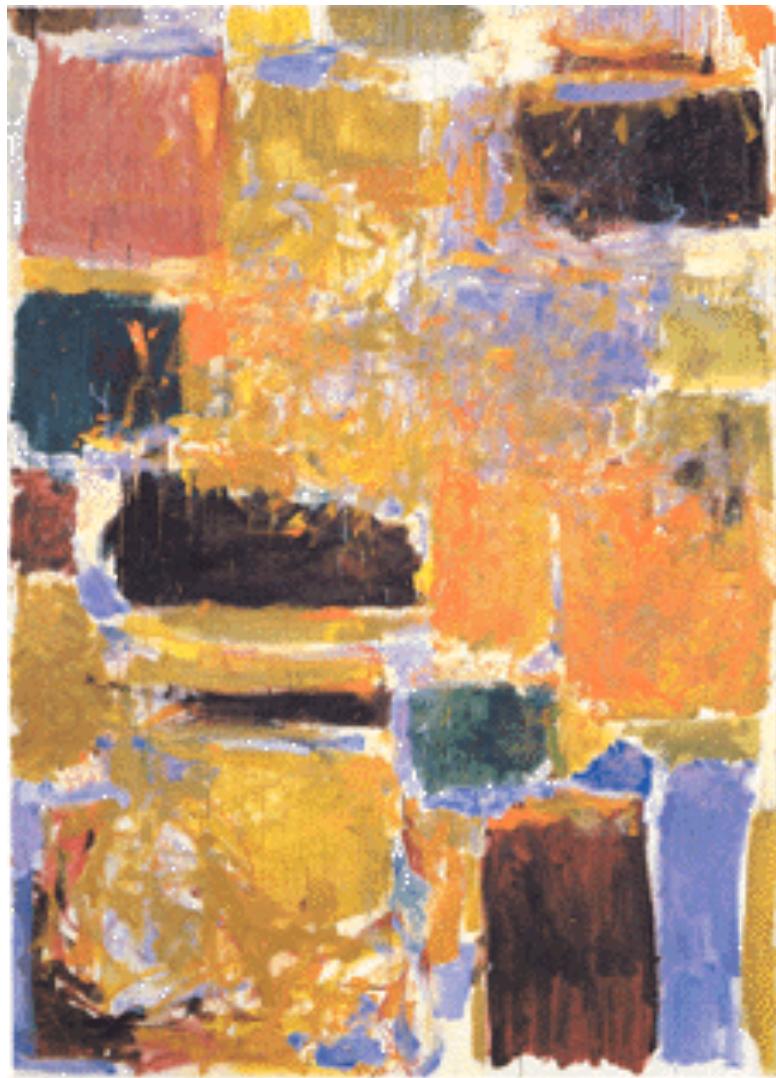
Instructors:
John Wawrynek and Vladimir Stojanovic

Lecture 18

Outline



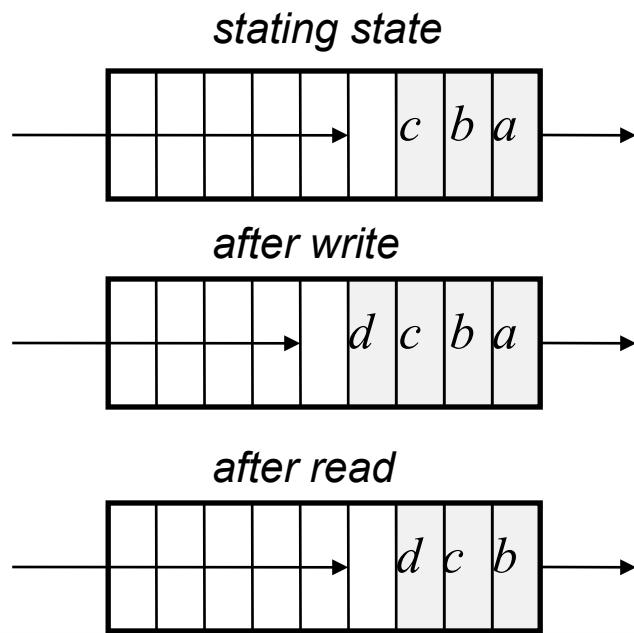
- FIFOs*
- Counters*
- LFSRs*



FIFOs

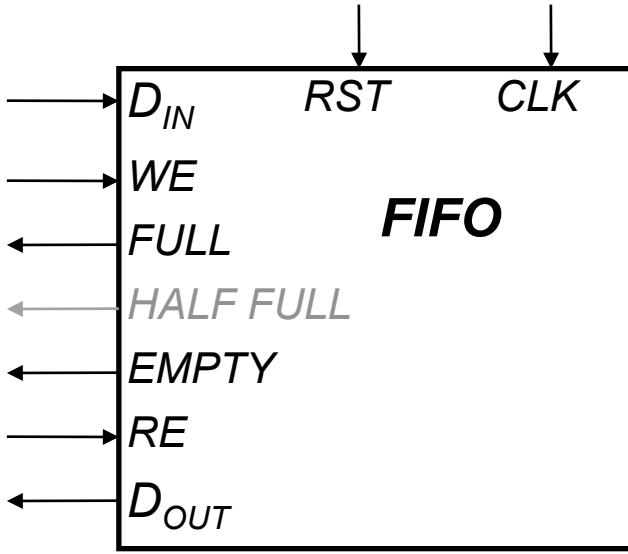
First-in-first-out (FIFO) Memory

- ❑ Used to implement *queues*.
- ❑ These find common use in computers and communication circuits.
- ❑ Generally, used to “decouple” actions of producer and consumer:



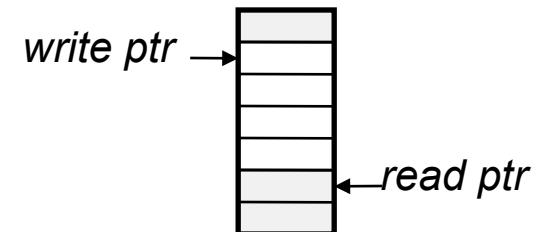
- *Producer can perform many writes without consumer performing any reads (or vis versa). However, because of finite buffer size, on average, need equal number of reads and writes.*
- *Typical uses:*
 - *interfacing I/O devices. Example network interface. Data bursts from network, then processor bursts to memory buffer (or reads one word at a time from interface). Operations not synchronized.*
 - *Example: Audio output. Processor produces output samples in bursts (during process swap-in time). Audio DAC clocks it out at constant sample rate.*

FIFO Interfaces

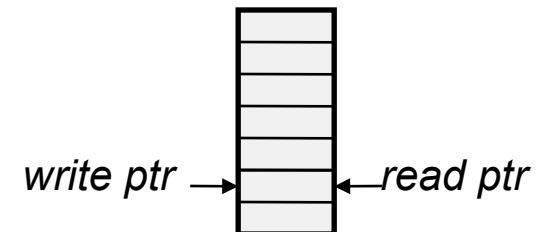


- ❑ After write or read operation, FULL and EMPTY indicate status of buffer.
- ❑ Used by external logic to control own reading from or writing to the buffer.
- ❑ FIFO resets to EMPTY state.
- ❑ HALF FULL (or other indicator of partial fullness) is optional.

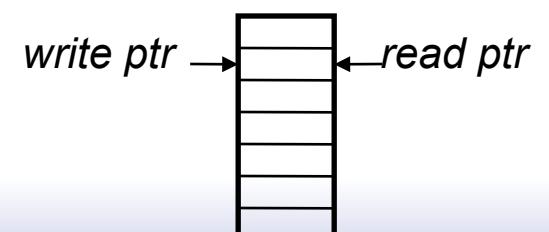
- Address pointers are used internally to keep next write position and next read position into a dual-port memory.



- If pointers equal after write \Rightarrow FULL:



- If pointers equal after read \Rightarrow EMPTY:



FIFO Implementation Details

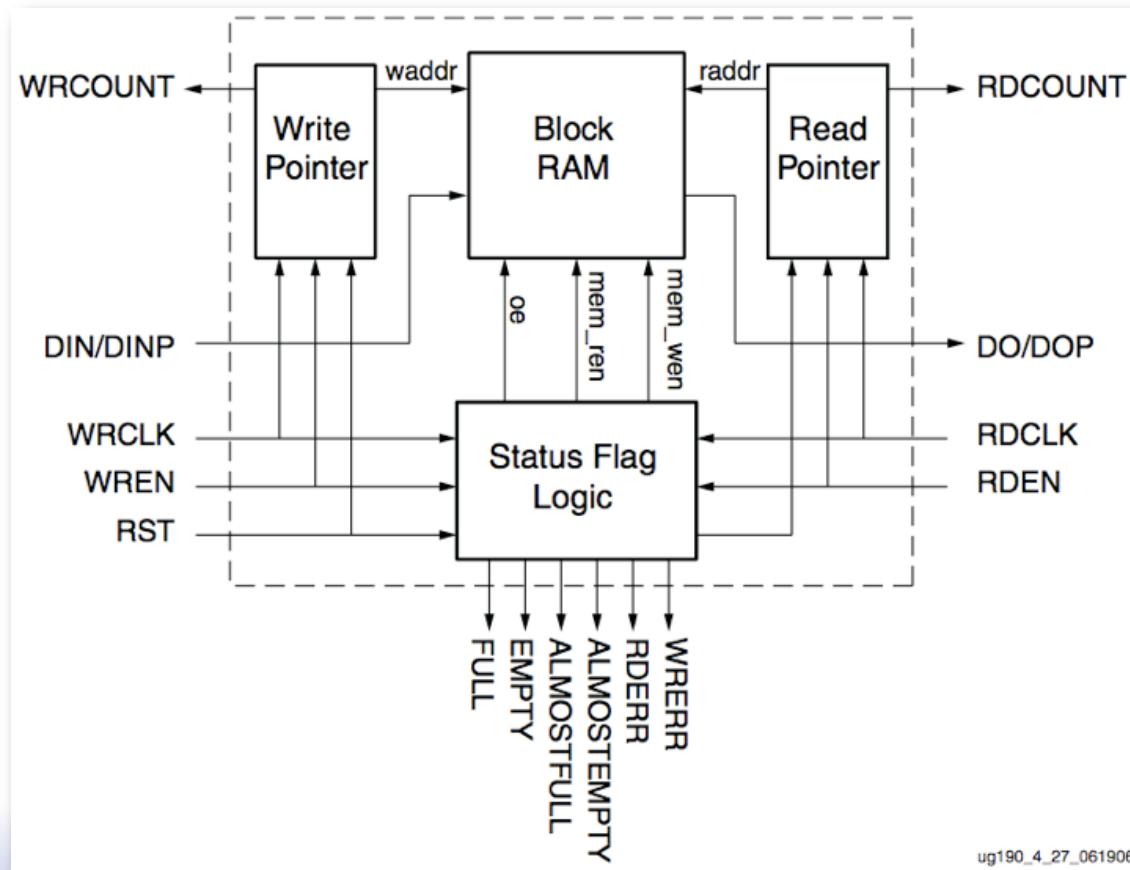
- Assume, dual-port memory with asynchronous read, synchronous write.
- Binary counter for each of read and write address. CEs (count enable) controlled by *WE* and *RE*.
- Equal comparator to see when pointers match.
- Flip-flop each for *FULL* and *EMPTY* flags:

WE	RE	equal	EMPTY_i	FULL_i
0	0	0	0	0
0	0	1	EMPTY_{i-1}	FULL_{i-1}
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	EMPTY_{i-1}	FULL_{i-1}

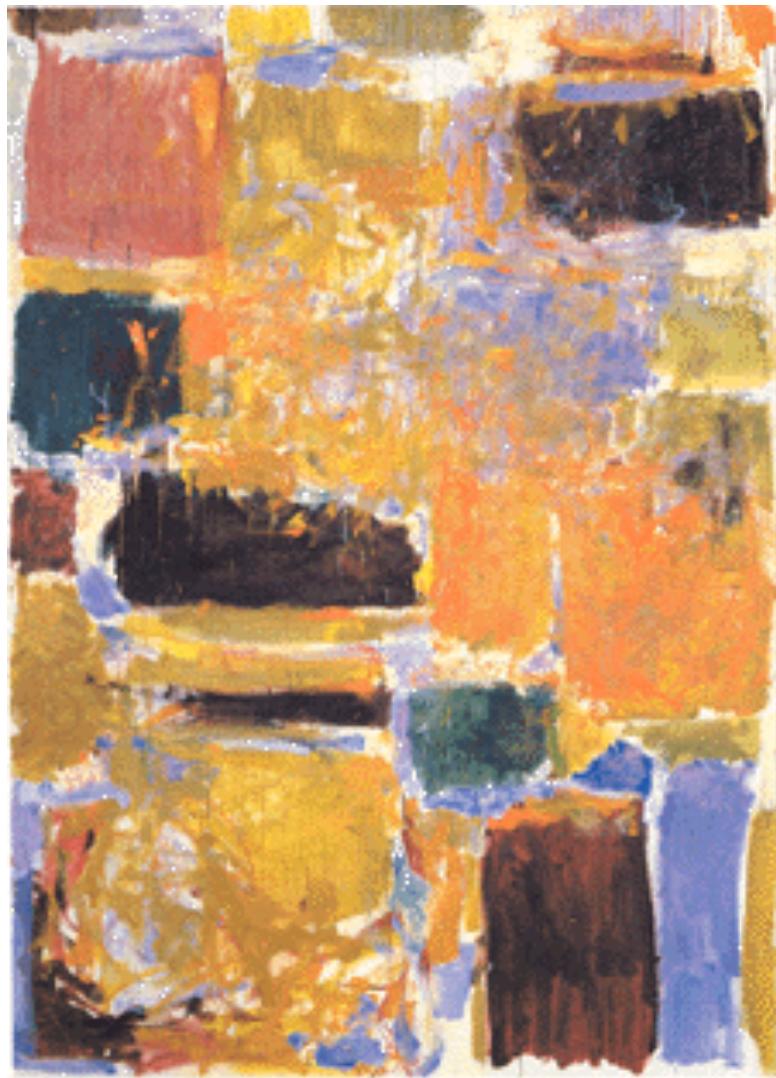
- Control logic (FSM) with truth-table shown to left.

Xilinx Virtex5 FIFOs

- ❑ Virtex5 BlockRAMS include dedicated circuits for FIFOs.
- ❑ Details in User Guide (ug190).
- ❑ Takes advantage of separate dual ports and independent ports clocks.



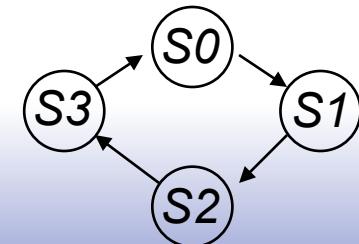
ug190_4_27_061906



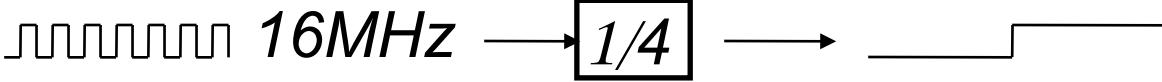
Counters

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:

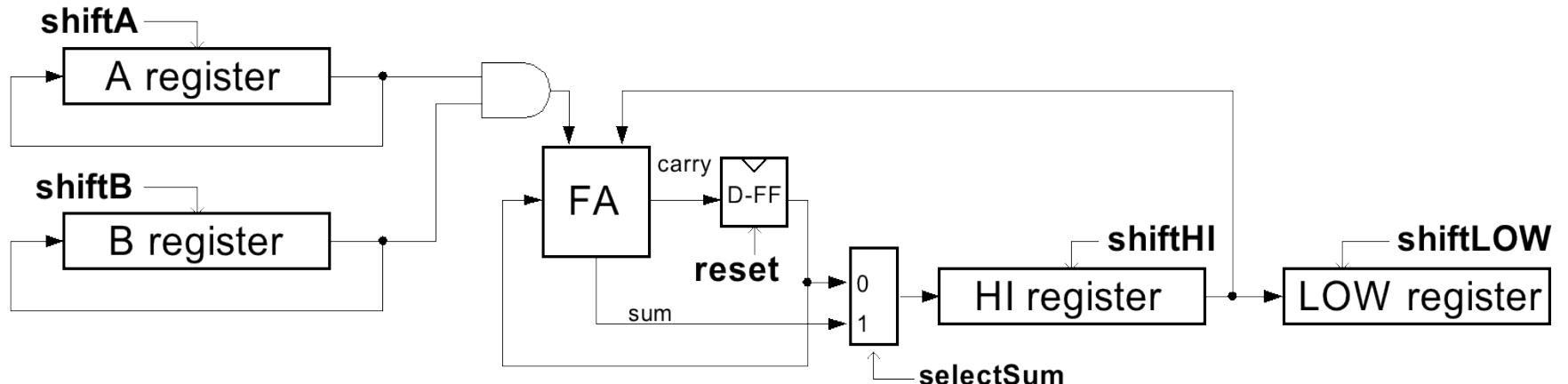


What are they used?

- 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):



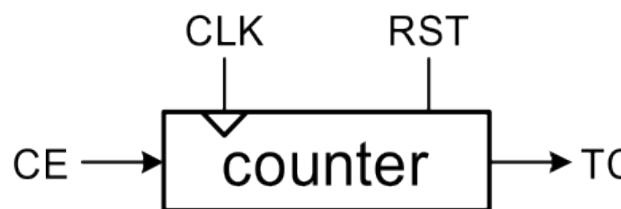
- Control Algorithm:

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

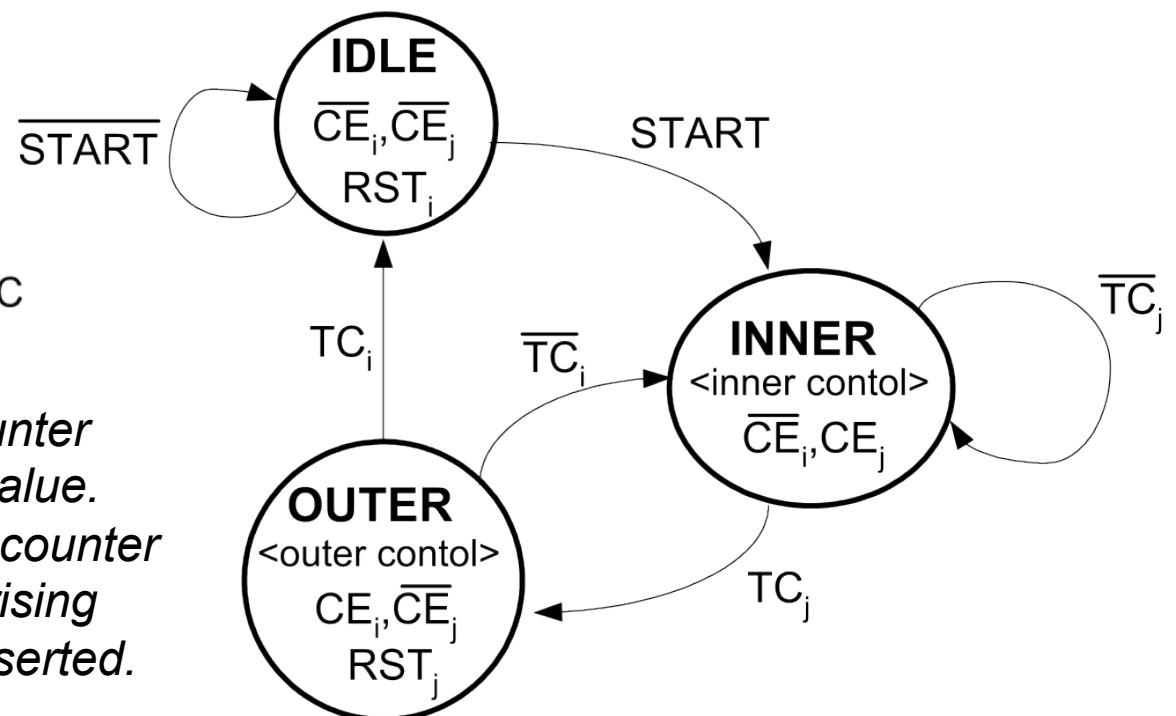
Note: The occurrence of a control signal x means $x=1$. The absence of x means $x=0$.

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.

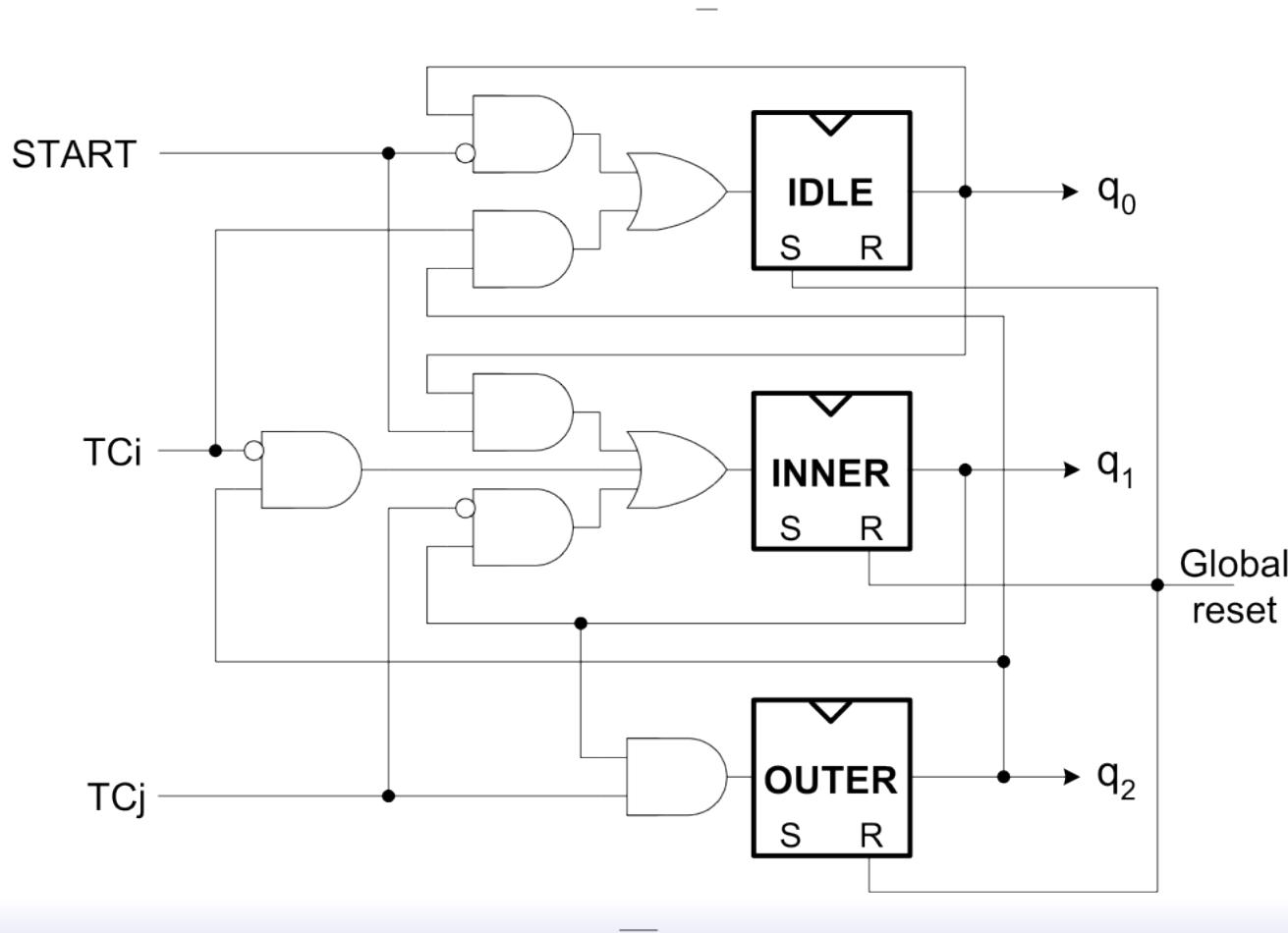


TC is asserted when the counter reaches its maximum count value. CE is “count enable”. The counter increments its value on the rising edge of the clock if CE is asserted.



Controller using Counters

- Controller circuit implementation:



- *Outputs:*

$$CE_i = q_2$$

$$CE_j = q_1$$

$$RST_i = q_0$$

$$RST_j = q_2$$

$$shiftA = q_1$$

$$shiftB = q_2$$

$$shiftLOW = q_2$$

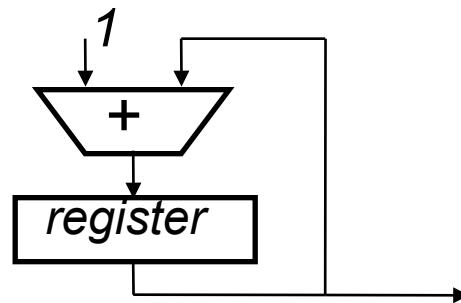
$$shiftHI = q_1 + q_2$$

$$reset = q_2$$

$$selectSUM = q_1$$

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.

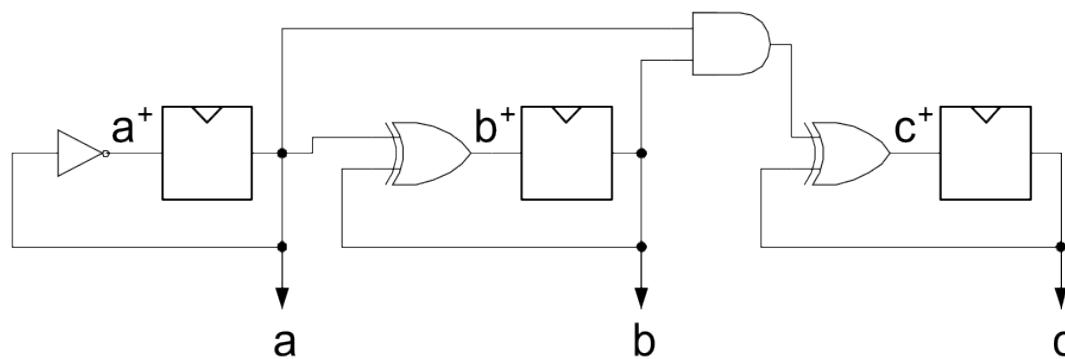
- Binary Counter Design:

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

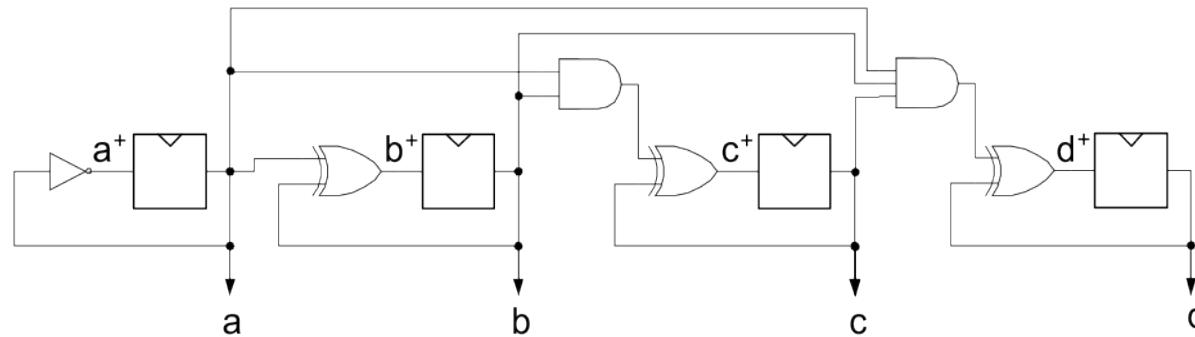
$$a^+ = a'$$
$$b^+ = a \oplus b$$

$$\begin{aligned} c^+ &= abc' + a'b'c + ab'c + \\ &\quad a'bc \\ &= a'c + abc' + b'c \\ &= c(a' + b') + c'(ab) \\ &= c(ab)' + c'(ab) \\ &= c \oplus ab \end{aligned}$$

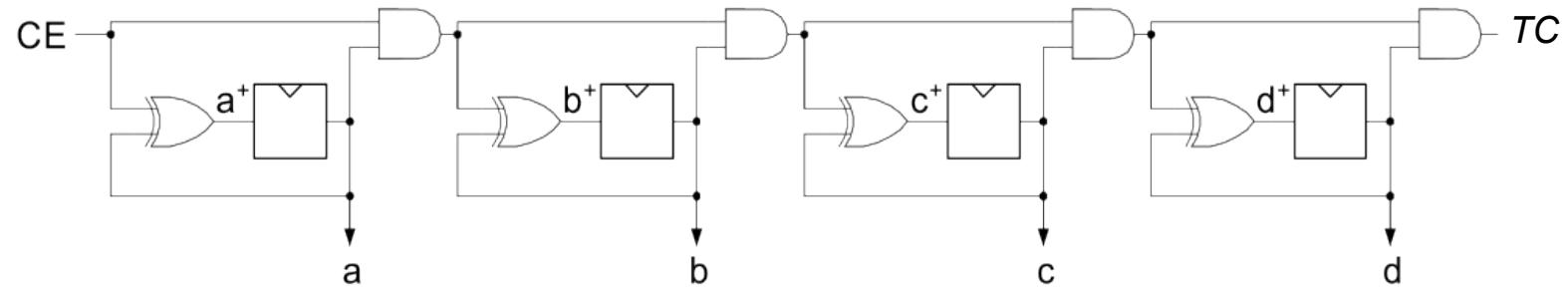


Synchronous Counters

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

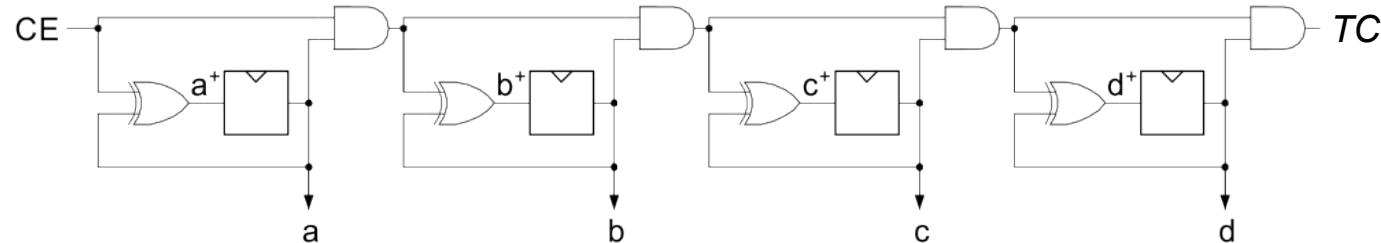


- Has difficulty scaling (AND gate inputs grow with n)

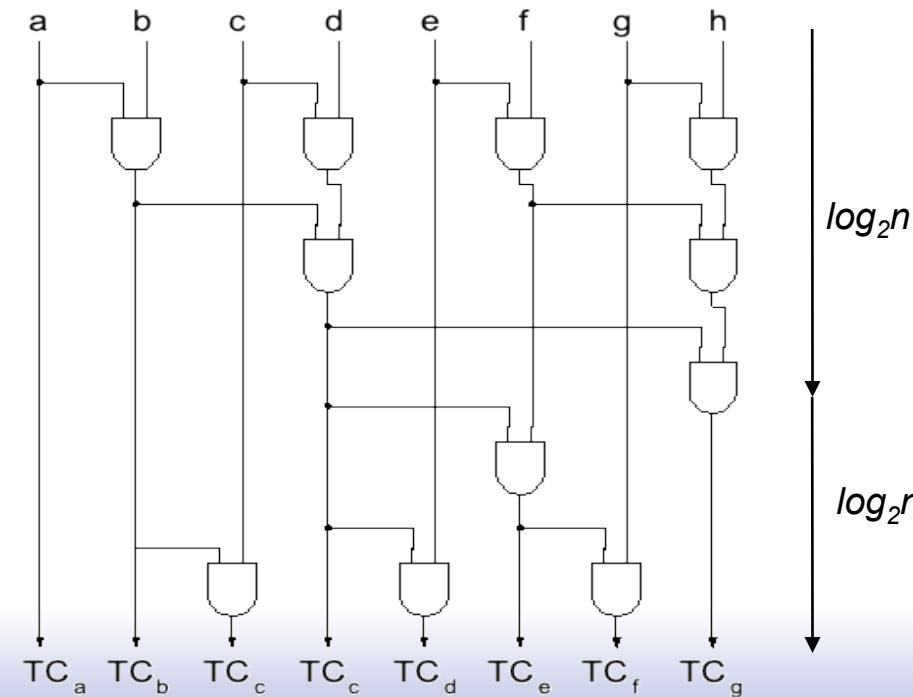


- CE is “count enable”, allows external control of counting,
- TC is “terminal count”, is asserted on highest value, allows cascading, external sensing of occurrence of max value.

Synchronous Counters



- How does this one scale?
- Generation of TC signals very similar to generation of carry signals in adder.
- “Parallel Prefix” circuit reduces delay:
- Delay grows αn



Up-Down Counter

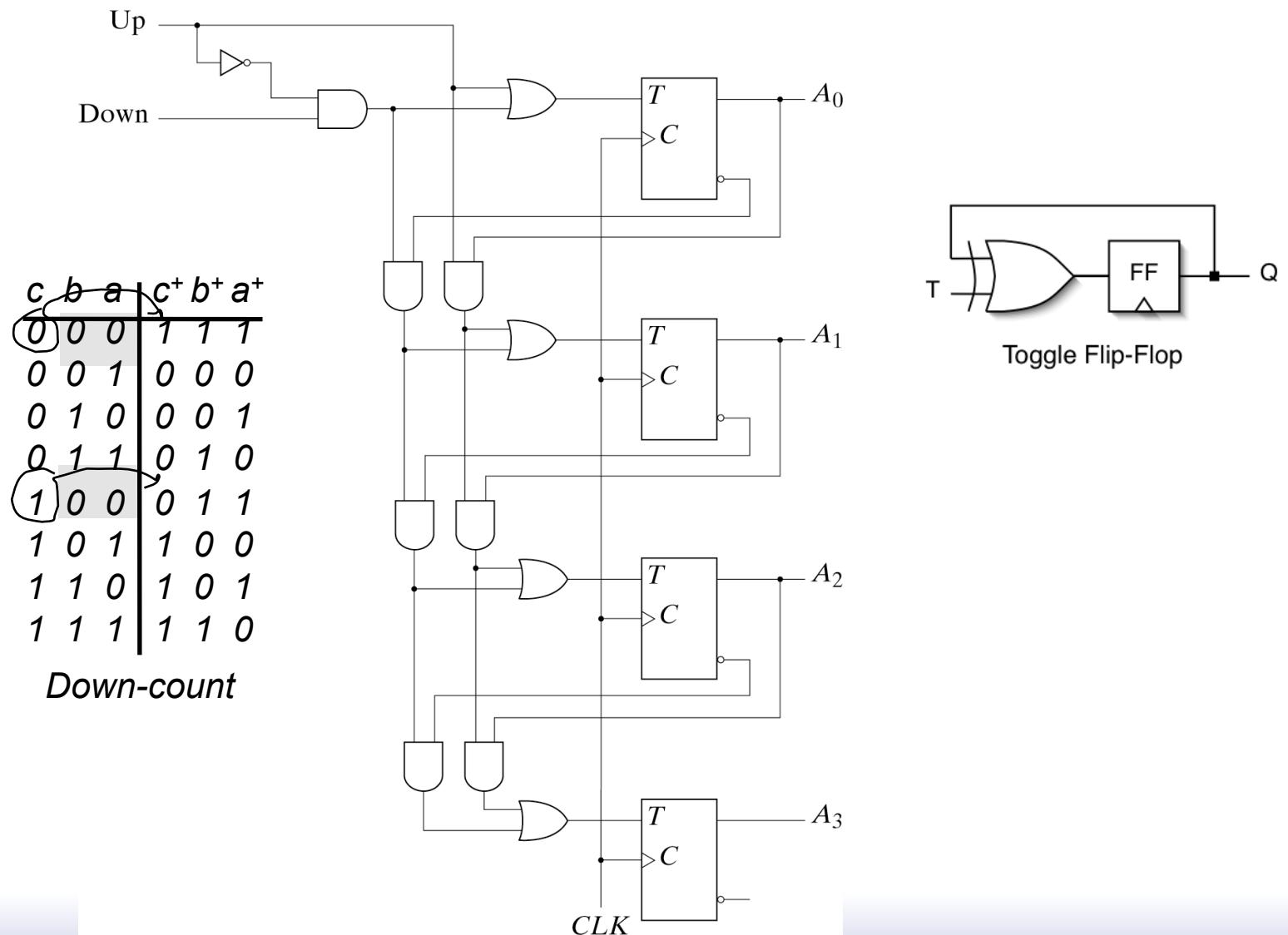
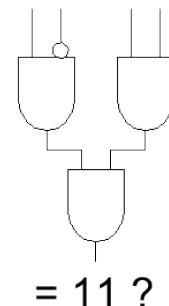
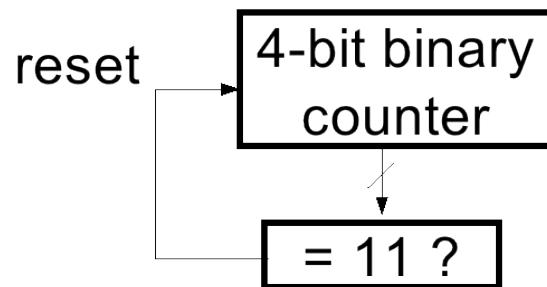


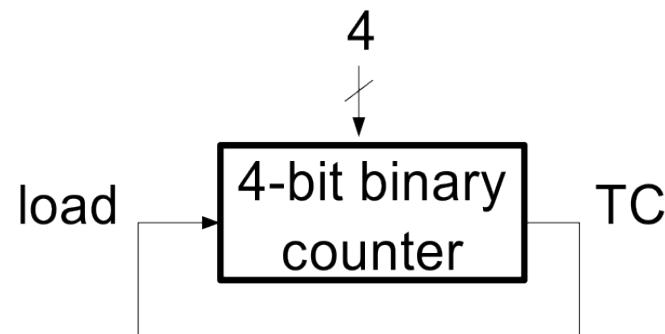
Fig. 6-13 4-Bit Up-Down Binary Counter

Odd Counts

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

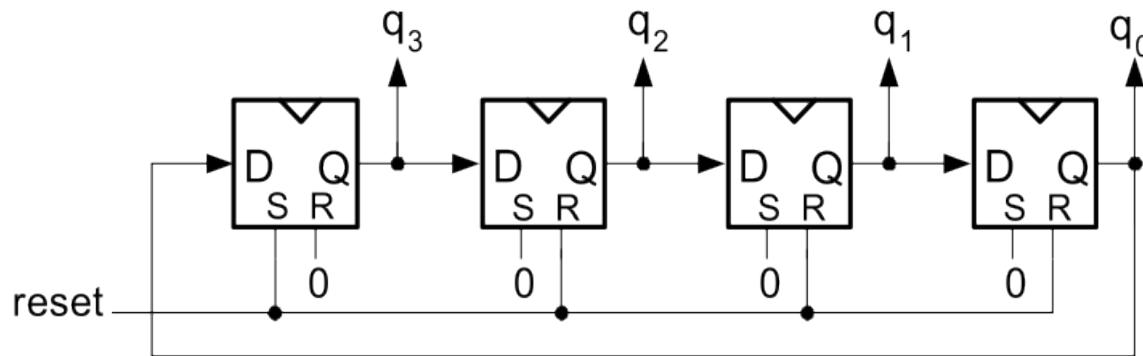


- *Alternative loadable counter:*

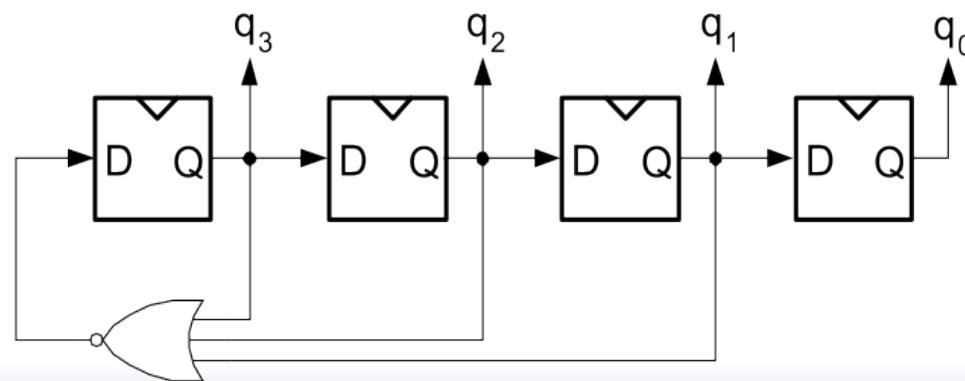


Ring Counters

- “one-hot” counters
0001, 0010, 0100, 1000, 0001, ...



“Self-starting” version:



“Ripple” counters

$A_3 A_2 A_1 A_0$

0000

0001

0010

0011

0100 time

0101

0110

0111

1000

1001

1010

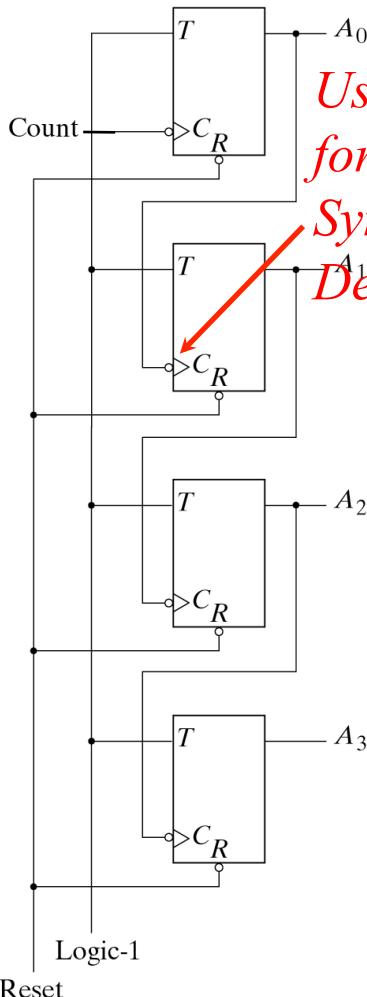
1011

1100

1101

1110

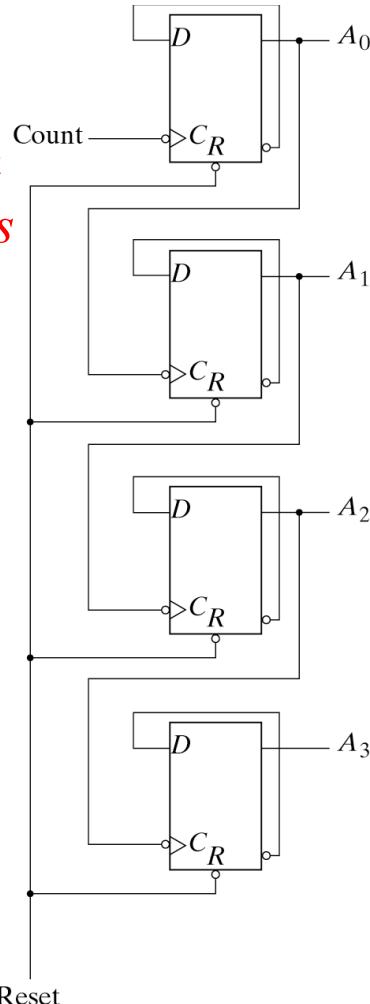
1111



(a) With T flip-flops

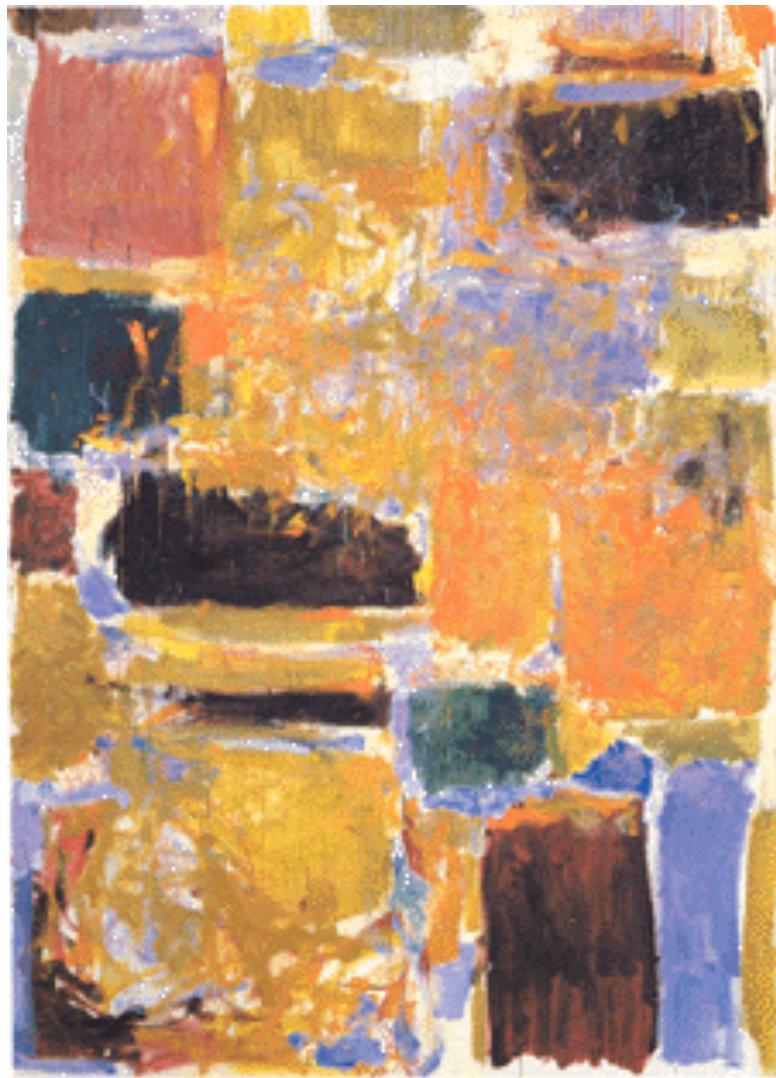
Fig. 6-8 4-Bit Binary Ripple Counter

Usually
forbidden in
Synchronous
Design



(b) With D flip-flops

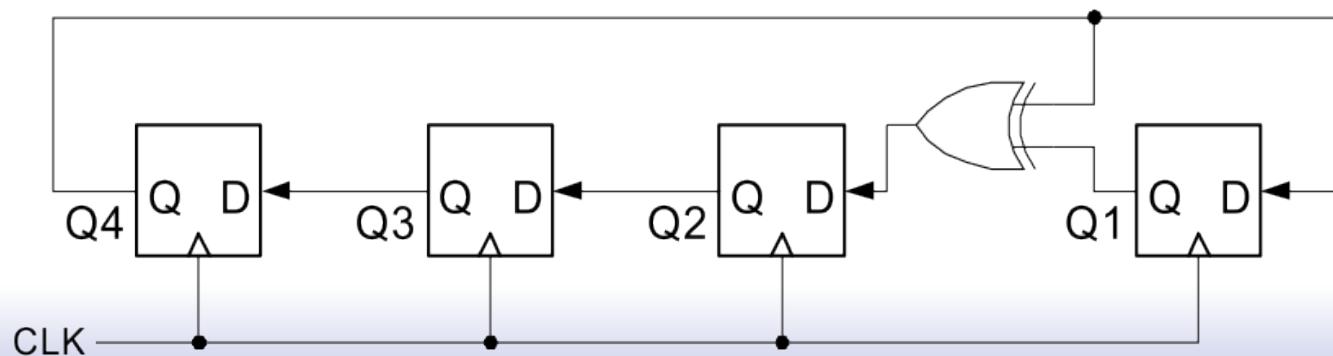
- Each stage is 1/2 of previous.
 - Look at output waveforms:
-
- Often called “asynchronous” counters.
 - A “T” flip-flop is a “toggle” flip-flop. Flips it state on cycles when $T=1$.



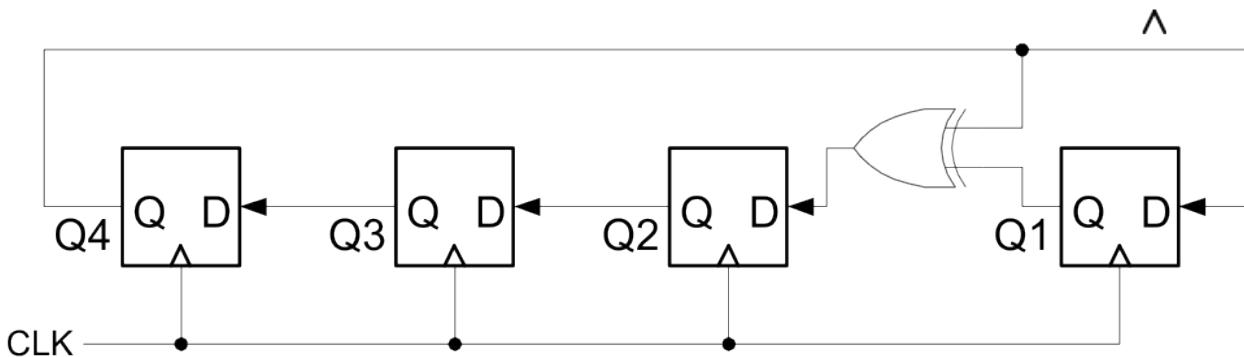
LFSRs

Linear Feedback Shift Registers (LFSRs)

- These are n-bit counters exhibiting pseudo-random behavior.
- Built from simple shift-registers with a small number of xor gates.
- Used for:
 - random number generation
 - counters
 - error checking and correction
- Advantages:
 - very little hardware
 - high speed operation
- Example 4-bit LFSR:



4-bit LFSR



- Circuit counts through $2^4 - 1$ different non-zero bit patterns.
- Leftmost bit decides whether the “10011” xor pattern is used to compute the next value or if the register just shifts left.
- Can build a similar circuit with any number of FFs, may need more xor gates.
- In general, with n flip-flops, $2^n - 1$ different non-zero bit patterns.
- (Intuitively, this is a counter that wraps around many times and in a strange way.)

	0001
	0010
	0100
	1000
xor	0011
	0110
xor	1000
	0011
xor	0110
	1100
xor	1011
	0111
xor	1010
	0111
xor	1110
	1111
xor	1101
	1001
xor	0001
	0101
xor	1011
	0101
xor	1111
	1111
xor	1101
	1001
xor	0001
	0101

Q4 Q3 Q2 Q1

Applications of LFSRs

□ Performance:

- In general, xors are only ever 2-input and never connect in series.
- Therefore the minimum clock period for these circuits is:

$$T > T_{\text{2-input-xor}} + \text{clock overhead}$$

- Very little latency, and independent of n!

□ This can be used as a fast counter, if the particular sequence of count values is not important.

- Example: micro-code micro-pc

- Can be used as a random number generator.
- Sequence is a pseudo-random sequence:
- numbers appear in a random sequence
- repeats every $2^n - 1$ patterns
- Random numbers useful in:
 - computer graphics
 - cryptography
 - automatic testing
 - Used for error detection and correction
 - CRC (cyclic redundancy codes)
 - Ethernet uses them

Galois Fields - the theory behind LFSRs

- LFSR circuits performs multiplication on a field.
- A field is defined as a set with the following:
 - two operations defined on it:
 - “addition” and “multiplication”
 - closed under these operations
 - associative and distributive laws hold
 - additive and multiplicative identity elements
 - additive inverse for every element
 - multiplicative inverse for every non-zero element
- *Example fields:*
 - set of rational numbers
 - set of real numbers
 - set of integers is not a field (why?)
- Finite fields are called Galois fields.
- *Example:*
 - Binary numbers 0,1 with XOR as “addition” and AND as “multiplication”.
 - Called GF(2).

Galois Fields - The theory behind LFSRs

- Consider polynomials whose coefficients come from GF(2).
- Each term of the form x^n is either present or absent.
- Examples: 0, 1, x , x^2 , and $x^7 + x^6 + 1$

$$= 1 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$$

- With addition and multiplication these form a field:
- “Add”: XOR each element individually with no carry:

$$\begin{array}{r} x^4 + x^3 + \quad + x + 1 \\ + \quad x^4 + \quad + x^2 + x \\ \hline x^3 + x^2 \quad + 1 \end{array}$$

- “Multiply”: multiplying by x^n is like shifting to the left.

$$\begin{array}{r} x^2 + x + 1 \\ \cdot \quad \quad x + 1 \\ \hline x^2 + x + 1 \\ \hline x^3 + x^2 + x \\ \hline x^3 \quad \quad \quad + 1 \end{array}$$

Galois Fields - The theory behind LFSRs

- These polynomials form a Galois (finite) field if we take the results of this multiplication modulo a prime polynomial $p(x)$.
 - A prime polynomial is one that cannot be written as the product of two non-trivial polynomials $q(x)r(x)$
 - Perform modulo operation by subtracting a (polynomial) multiple of $p(x)$ from the result. If the multiple is 1, this corresponds to XOR-ing the result with $p(x)$.
- For any degree, there exists at least one prime polynomial.
- With it we can form $GF(2^n)$
- Additionally, ...
- Every Galois field has a primitive element, α , such that all non-zero elements of the field can be expressed as a power of α . By raising α to powers (modulo $p(x)$), all non-zero field elements can be formed.
- Certain choices of $p(x)$ make the simple polynomial x the primitive element. These polynomials are called primitive, and one exists for every degree.
- For example, $x^4 + x + 1$ is primitive. So $\alpha = x$ is a primitive element and successive powers of α will generate all non-zero elements of $GF(16)$. Example on next slide.

Galois Fields - The theory behind LFSRs

$$\begin{aligned}\alpha^0 &= & 1 \\ \alpha^1 &= & x \\ \alpha^2 &= & x^2 \\ \alpha^3 &= & x^3 \\ \alpha^4 &= & x + 1 \\ \alpha^5 &= & x^2 + x \\ \alpha^6 &= & x^3 + x^2 \\ \alpha^7 &= & x^3 + x + 1 \\ \alpha^8 &= & x^2 + 1 \\ \alpha^9 &= & x^3 + x \\ \alpha^{10} &= & x^2 + x + 1 \\ \alpha^{11} &= & x^3 + x^2 + x \\ \alpha^{12} &= & x^3 + x^2 + x + 1 \\ \alpha^{13} &= & x^3 + x^2 + 1 \\ \alpha^{14} &= & x^3 + 1 \\ \alpha^{15} &= & 1\end{aligned}$$

- Note this pattern of coefficients matches the bits from our 4-bit LFSR example.

$$\begin{aligned}\alpha^4 &= x^4 \bmod x^4 + x + 1 \\ &= x^4 \text{ xor } x^4 + x + 1 \\ &= x + 1\end{aligned}$$

- In general finding primitive polynomials is difficult. Most people just look them up in a table, such as:

Primitive Polynomials

$x^2 + x + 1$
 $x^3 + x + 1$
 $x^4 + x + 1$
 $x^5 + x^2 + 1$
 $x^6 + x + 1$
 $x^7 + x^3 + 1$
 $x^8 + x^4 + x^3 + x^2 + 1$
 $x^9 + x^4 + 1$
 $x^{10} + x^3 + 1$
 $x^{11} + x^2 + 1$

$x^{12} + x^6 + x^4 + x + 1$
 $x^{13} + x^4 + x^3 + x + 1$
 $x^{14} + x^{10} + x^6 + x + 1$
 $x^{15} + x + 1$
 $x^{16} + x^{12} + x^3 + x + 1$
 $x^{17} + x^3 + 1$
 $x^{18} + x^7 + 1$
 $x^{19} + x^5 + x^2 + x + 1$
 $x^{20} + x^3 + 1$
 $x^{21} + x^2 + 1$

$x^{22} + x + 1$
 $x^{23} + x^5 + 1$
 $x^{24} + x^7 + x^2 + x + 1$
 $x^{25} + x^3 + 1$
 $x^{26} + x^6 + x^2 + x + 1$
 $x^{27} + x^5 + x^2 + x + 1$
 $x^{28} + x^3 + 1$
 $x^{29} + x + 1$
 $x^{30} + x^6 + x^4 + x + 1$
 $x^{31} + x^3 + 1$
 $x^{32} + x^7 + x^6 + x^2 + 1$

Galois Field

Multiplication by x

Hardware

\Leftrightarrow shift left

Taking the result mod $p(x)$

\Leftrightarrow XOR-ing with the coefficients of $p(x)$

when the most significant coefficient is 1.

Obtaining all $2^n - 1$ non-zero elements by evaluating x^k
for $k = 1, \dots, 2^n - 1$

\Leftrightarrow Shifting and XOR-ing $2^n - 1$ times.

Building an LFSR from a Primitive Polynomial

- For k-bit LFSR number the flip-flops with FF1 on the right.
- The feedback path comes from the Q output of the leftmost FF.
- Find the primitive polynomial of the form $x^k + \dots + 1$.
- The $x^0 = 1$ term corresponds to connecting the feedback directly to the D input of FF 1.
- Each term of the form x^n corresponds to connecting an xor between FF n and $n+1$.
- 4-bit example, uses $x^4 + x + 1$
 - $x^4 \Leftrightarrow$ FF4's Q output
 - $x \Leftrightarrow$ xor between FF1 and FF2
 - $1 \Leftrightarrow$ FF1's D input
- To build an 8-bit LFSR, use the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ and connect xors between FF2 and FF3, FF3 and FF4, and FF4 and FF5.

