



# Lecture Outline

---

## Reminders to self:

- ☐ Turn on lecture recording to Cloud
- ☐ Turn on Zoom microphone

- Last Lecture

- Reviewed HW 9-1
  - Continued Flip-Flops: finished D, then S-R, J-K and T flip-flops
  - Asynchronous clear and preset, and clock enable flip-flop inputs

- Today's Lecture

- Registers
    - Basic register
    - Data transfer between registers
    - Parallel adder with accumulator register
    - Start shift registers (if there is time)



# Handouts and Announcements

---

- Announcements
  - Homework Problem 11-3
    - Posted on Carmen Saturday (2/25)
    - Due: 11:59pm Thursday 3/2
  - Homework Reminder: HW 9-4
    - HW 9-4 now past due
    - HW 11-1 and 11-2 due 11:59pm Tuesday 2/28
  - Participation Quiz 9
    - Available 12:25pm, Due Tuesday at 12:25pm
    - Available another 24 hr with late penalty
  - Read for Wednesday: pages 384-395



## Participation Quiz 9: 4 questions

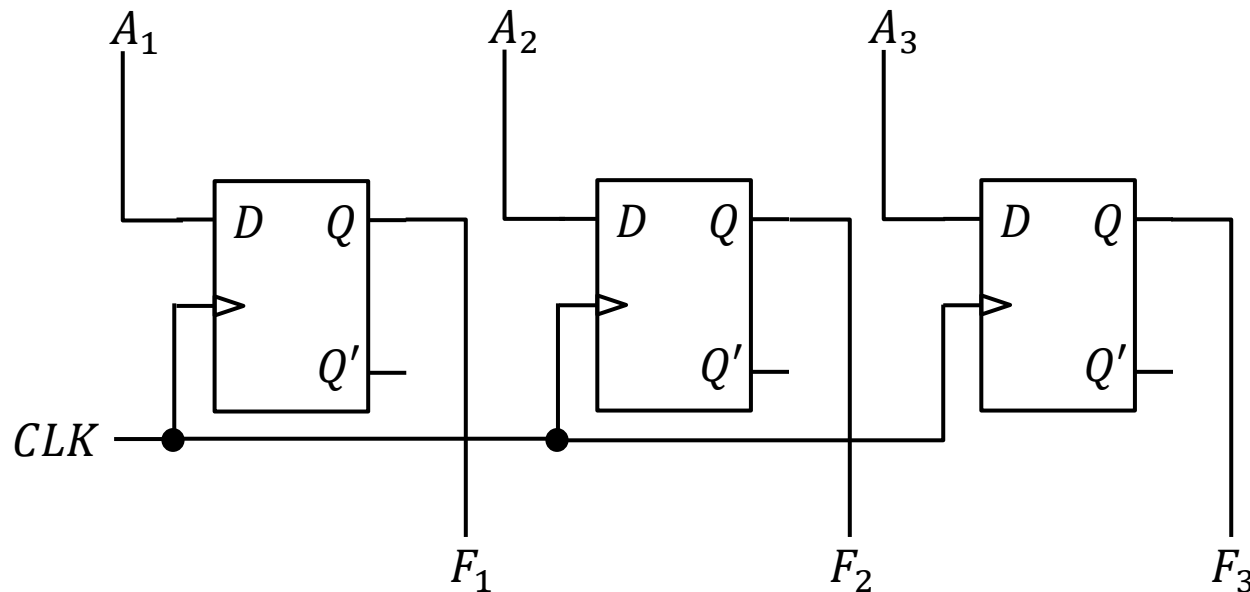
---

- In all of these questions the letters of the inputs have been changed to generic  $A$  and  $B$  to protect the identity of the Flip-Flops.
  - All of the questions also ask “*The equation describes the next state of the output of the flip-flop. Which type of flip-flop is it?*” – with different equations.
1.  $Q^+ = A$  (Choices: D, S-R, J-K, or T Flip-Flop)
  2.  $Q^+ = AQ' + A'Q$  (Choices: D, S-R, J-K, or T Flip-Flop)
  3.  $Q^+ = A + B'Q$  and require ( $AB = 0$ ) (Choices: D, S-R, J-K, or T Flip-Flop)
  4.  $Q^+ = AQ' + B'Q$  (Choices: D, S-R, J-K, or T Flip-Flop)



## D Flip-Flop & Registers

- D-type Flip-Flops are often used in “Registers”
- Typically the simplest choice for implementing Registers
- Register  $\Rightarrow$  An array of related bits
- Example: 3-bit register using D Flip-Flops





# T Flip-Flop & Counters

- T-type Flip-Flops are often used to build “Counters”
- Typically the simplest choice for implementing counters
- Count number of rising (or falling) edges of pulses that hit the clock inputs of an array of T Flip-Flops
- Toggle bits in appropriate order
- If after 1001 the counter returns to 0000, what type of counter is this? BCD
- Note: Although T flip-flops are typically the simplest choice for implementing counters, other types of flip-flops can be used.
- And I will probably assign homework requiring you to use other types of Flip-Flops. Two reasons:
  1. Practice implementing operations with other flip-flop types
  2. To more fully exercise techniques for counter design as you are learning them

$Q_3$	$Q_2$	$Q_1$	$Q_0$
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1

BCD:  
Binary  
Coded  
Decimal  
counter

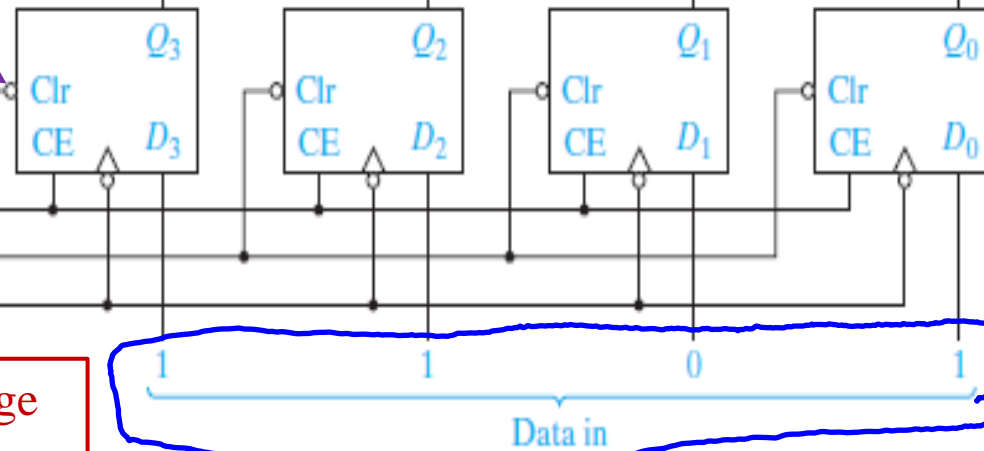


# Registers

- A register consists of a group of flip-flops with a *common clock input*
- Registers are commonly used to *store* or *shift* binary data
- Example: 4-Bit D Flip-Flop Register with **Data**, Load, Clear and Clock Inputs

Active-low clear

On  $Clk \downarrow$  when  $\overbrace{Data\ out}^{0 \rightarrow 1} Load = CE = 1$   
 $0 \rightarrow 1$        $0 \rightarrow 1$        $0 \rightarrow 0$        $0 \rightarrow 1$

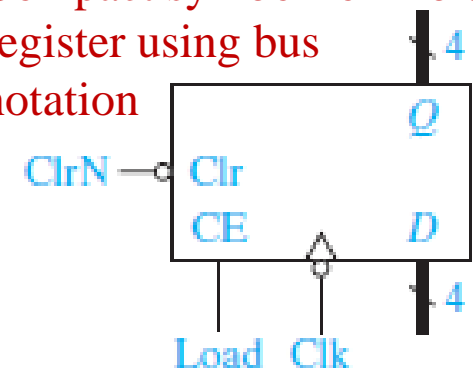


Falling-edge clock

(b) With clock enable

Clock enable controls whether or not data saved on next clock falling edge

Compact symbol for 4-bit register using bus notation

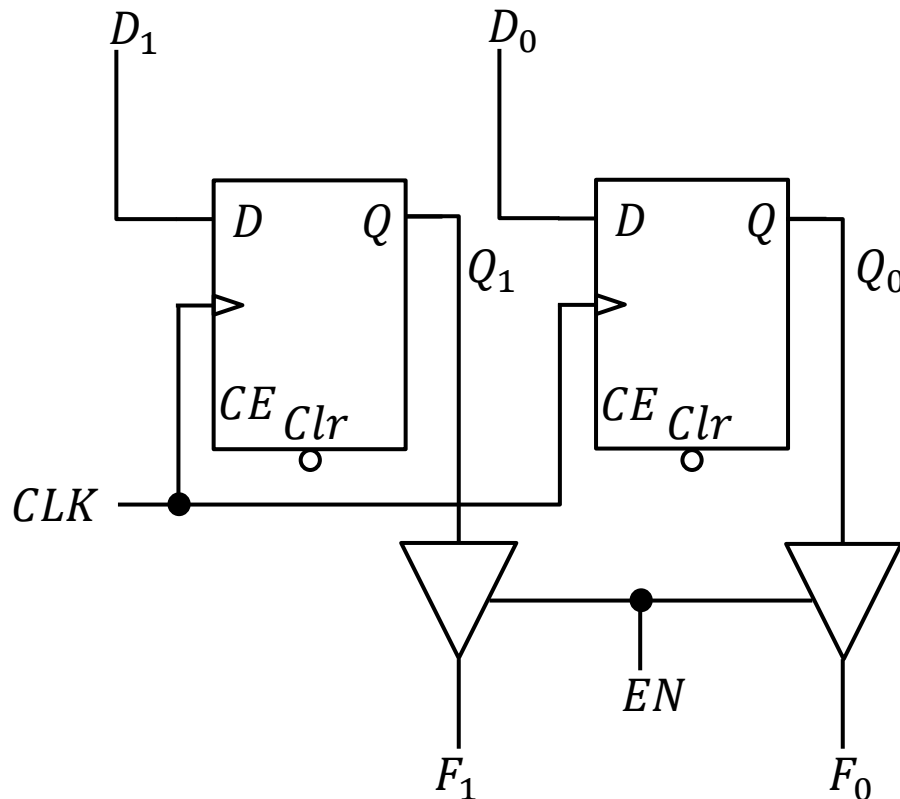


(c) Symbol



# Registers

- Register with *Read Enable* : 2-bit used to show concept



$$EN = 1 \rightarrow \begin{cases} F_0 = Q_0 \\ F_1 = Q_1 \end{cases}$$

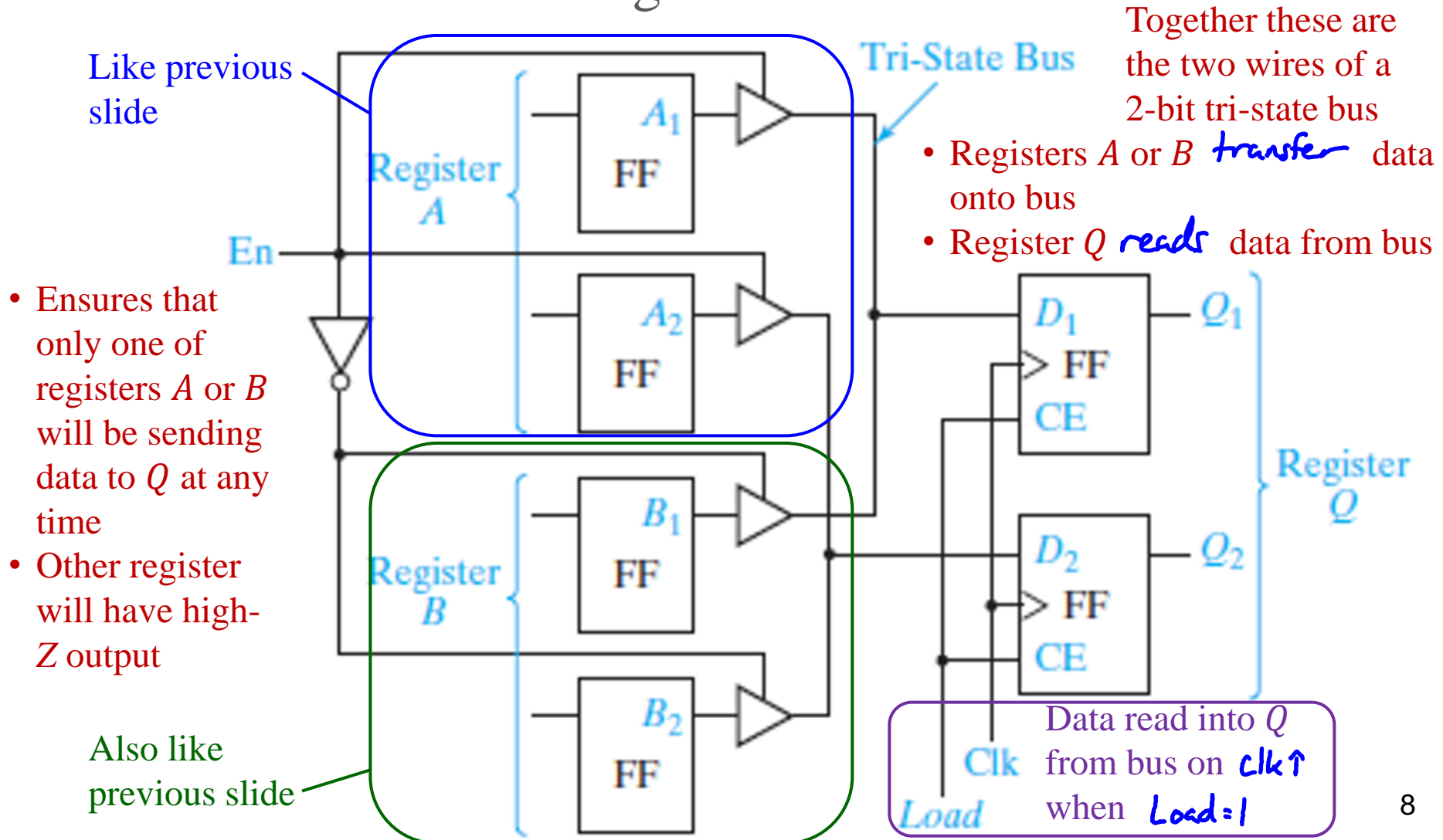
$$EN = 0 \rightarrow \begin{cases} F_0 = Z \\ F_1 = Z \end{cases}$$

Availability of high-Z output state allows outputs from *multiple registers* to feed into a common *data bus*



# Registers

## • Data Transfer between registers

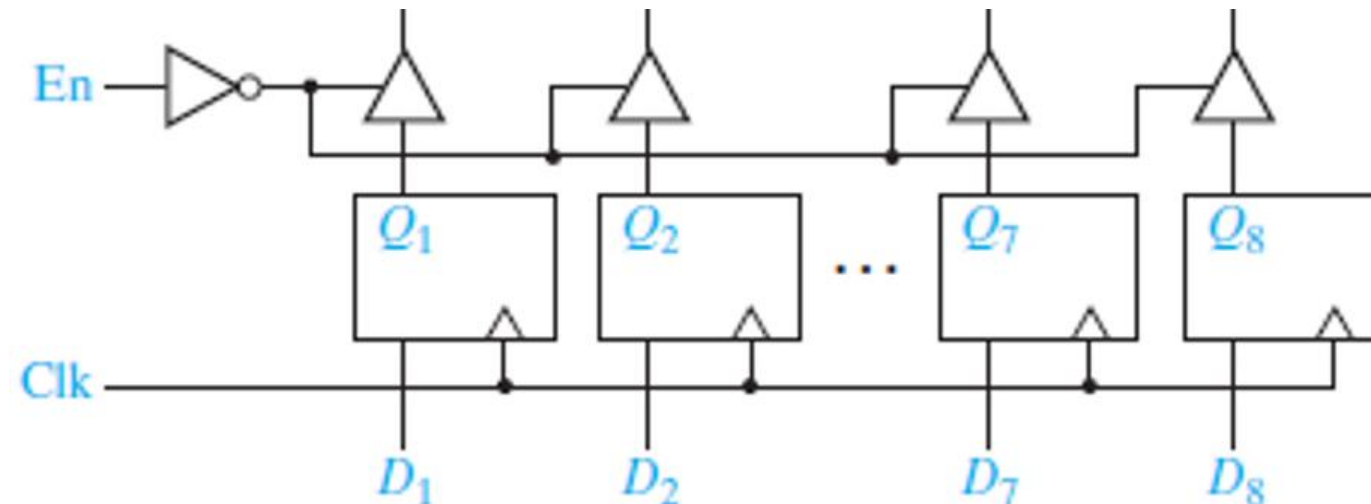




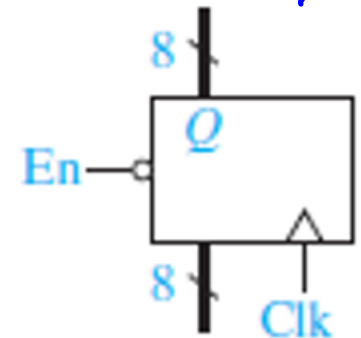


# Registers

- The concepts on previous slide, with two 2-bit registers, extend to:
  - **Registers with larger numbers of bits (8 bit example shown below)**
    - Tri-state buffers at flip-flop outputs
    - The buffers enabled when  $En = 0$
  - **More than two registers sending to the bus**
    - Only one register may have active output at a time
    - Bus controller to make sure other registers have high-Z outputs
    - Simple inverter on  $En$  not sufficient with more than two registers writing to bus



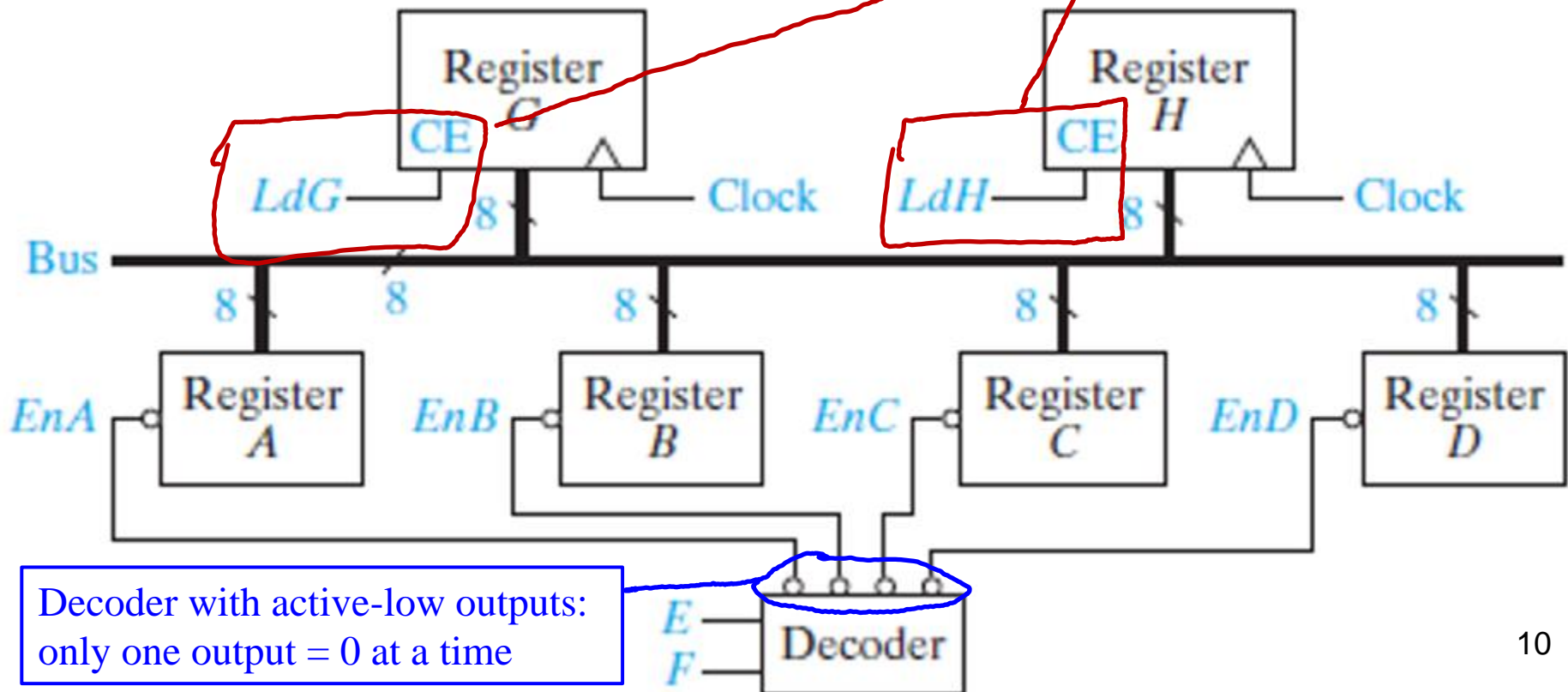
Compact symbol for 8-bit register with tri-state outputs, using bus notation. Note “ $En$ ” rather than “ $CE$ ” (but could have both)





# Registers

- More than two registers sending to the bus (four shown below)
- Only one 8-bit register may have active output at a time
- 2-to-4 decoder (introduced in Chapter 9) used as bus controller to select active register
- E.g.  $EF = 00$ :  $A \rightarrow \text{bus}$ ;  $EF = 01$ :  $B \rightarrow \text{bus}$ ;  $EF = 10$ :  $C \rightarrow \text{bus}$ ;  $EF = 11$ :  $D \rightarrow \text{bus}$
- Other registers' outputs not enabled  $\Rightarrow$  have high- $z$  outputs
- Data loaded into Register  $G$  and/or  $H$  depending on  $LdG$  and  $LdH$  values when  $Clock \uparrow$



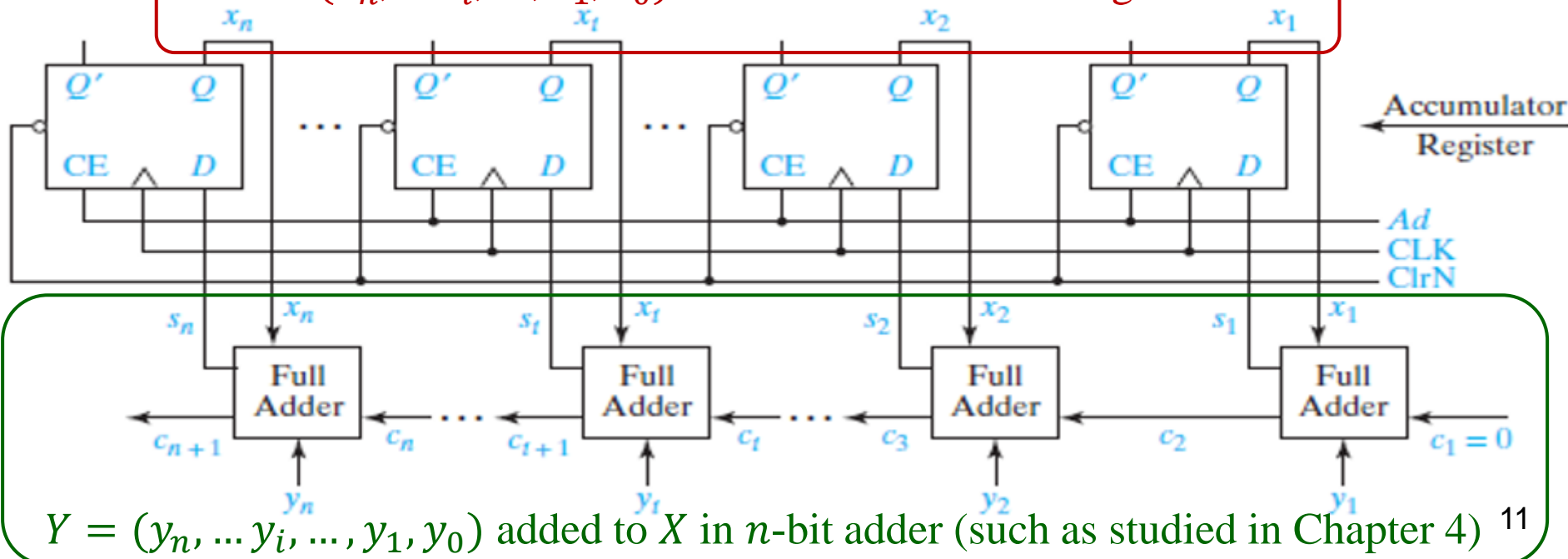


## Parallel Adder with Accumulator

- In computer arithmetic circuits:

1. Frequently desirable to store one number in a register of flip-flops (called an accumulator)
2. Add a second number to it
3. Store final result in the accumulator

$X = (x_n, \dots, x_i, \dots, x_1, x_0)$  stored in accumulator register

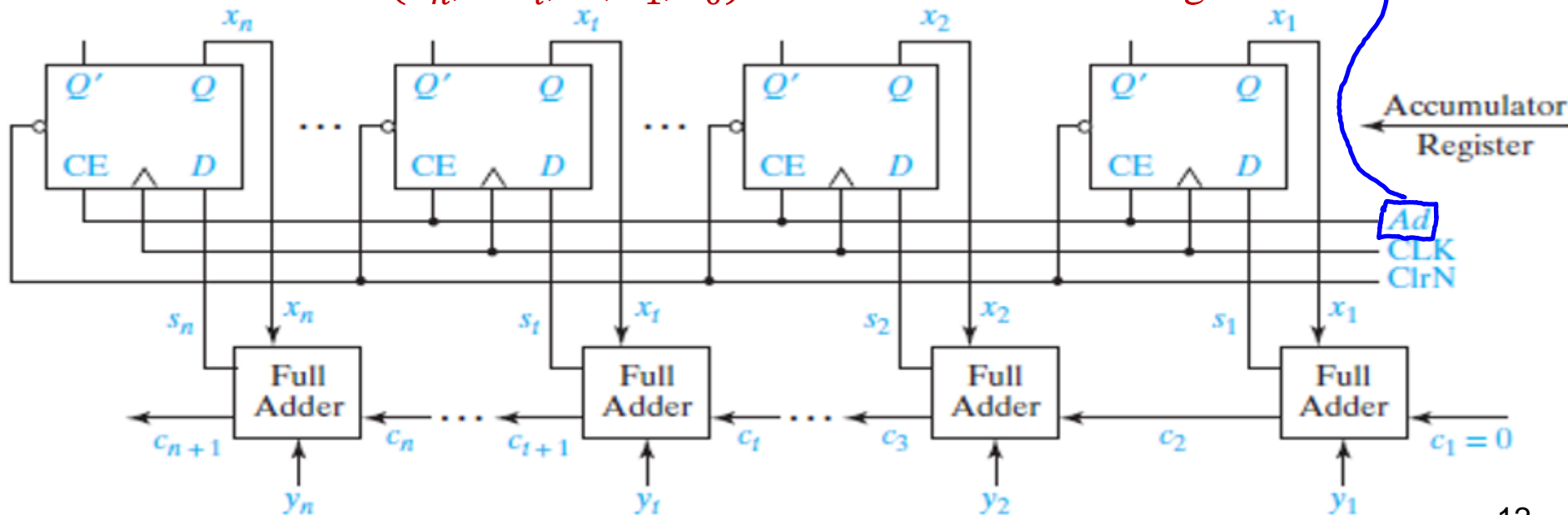




## Parallel Adder with Accumulator

- After addition is complete (including carries propagated through)
- Make  $Ad = 1$  to transfer sum  $S = (s_n, \dots, s_i, \dots, s_1, s_0)$  into accumulator (replaces  $X$ ) at next  $CLK \uparrow$
- Note that adder with accumulator has  $n$  cells of Full Adder + D Flip-Flop

$X = (x_n, \dots, x_i, \dots, x_1, x_0)$  stored in accumulator register

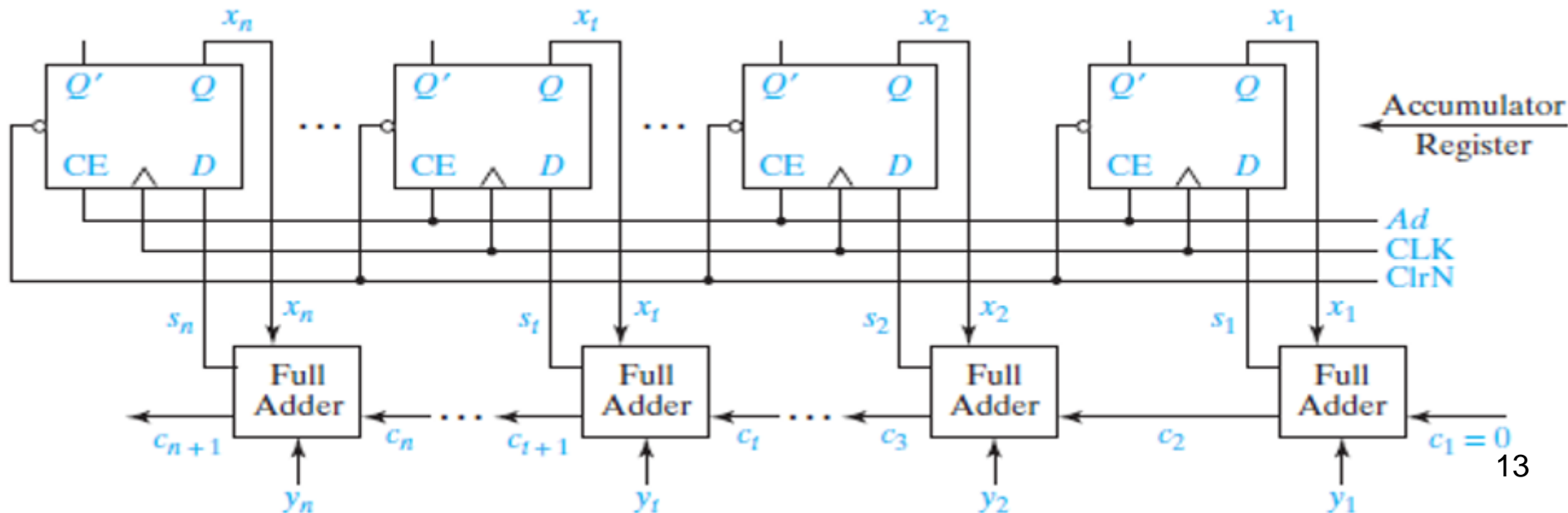


$Y = (y_n, \dots, y_i, \dots, y_1, y_0)$  added to  $X$  in  $n$ -bit adder (such as studied in Chapter 4)




## Parallel Adder with Accumulator

- First value of  $X$  has to be loaded into accumulator before addition can take place
- One approach
  1. Use  $ClrN$  to make all bits of  $X$  to 0
  2. Put  $X$  data in  $Y$  input
  3. Add to 0 in accumulator
- Done with existing hardware, but more clock cycles needed
- Another approach: MUX at accumulator inputs to select either first data or sum



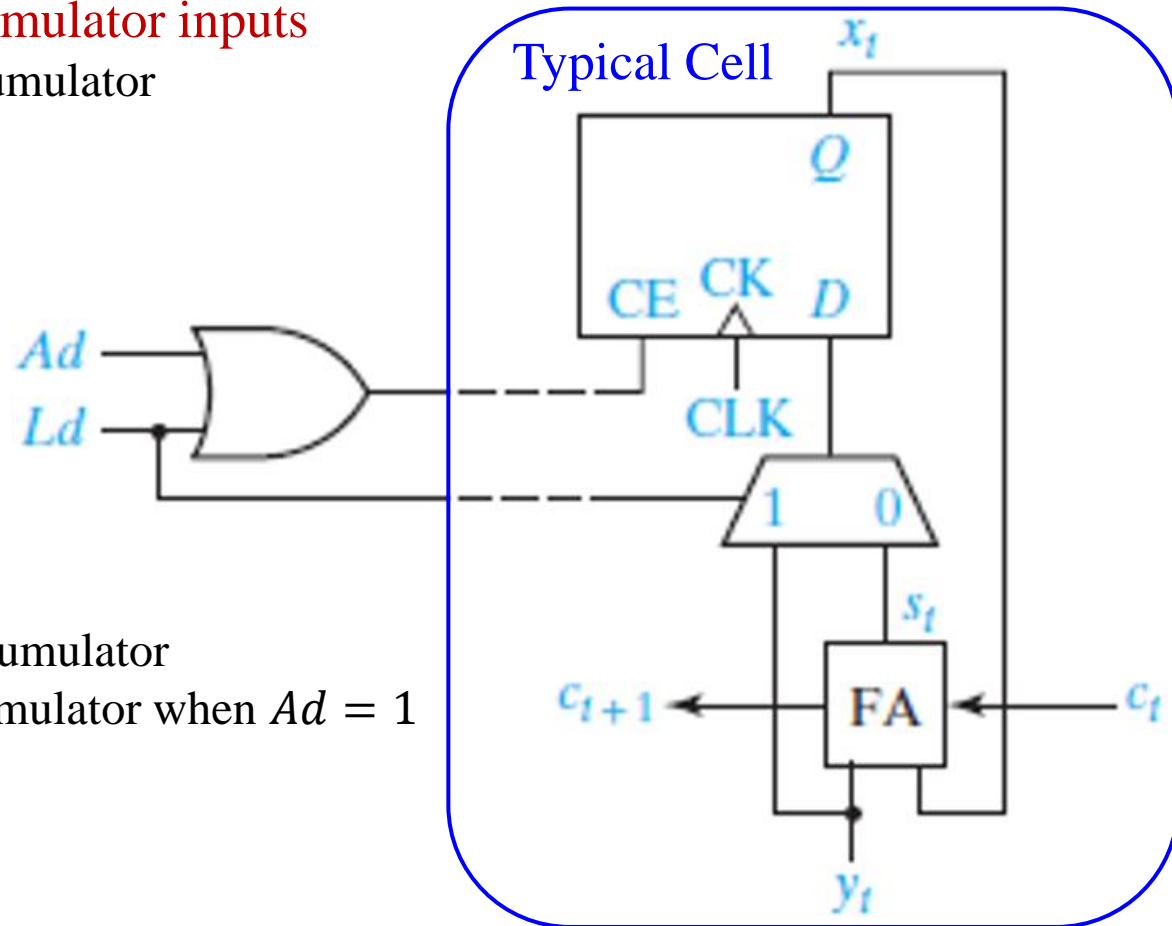
## Registers

## Parallel Adder with Accumulator

- Another approach: MUX at accumulator inputs to select either first data or sum
  - Typical cell with MUX at accumulator inputs
    - Eliminates steps to clear accumulator
    - But more complex hardware
- 
- The diagram, titled "Typical Cell", illustrates a hardware configuration for an accumulator. It features a multiplexer (MUX) with two inputs. The top input is labeled  $x_1$  (the first data input), and the bottom input is labeled  $Q$  (the sum output). The MUX is controlled by a signal  $Q$  (the sum output) and its output is connected to the accumulator input. This setup allows the accumulator to either add the first data input  $x_1$  or clear itself by selecting the zero output of the MUX.

## Load control:

- $Ld = 1$  to save first data into accumulator
- $Ld = 0$  to transfer sum into accumulator when  $Ad = 1$





# Registers

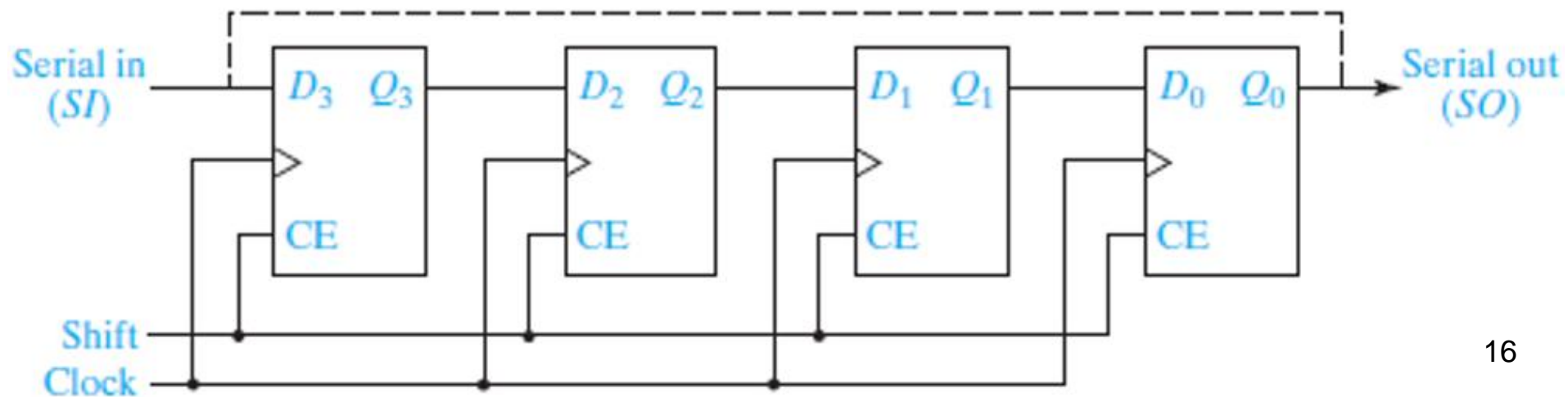
---

- Our registers so far have been Parallel Load
  - All bits updated with new data at the same clock edge
- Registers with serial input/output (I/O) are also possible and useful
- Pure serial:
  - Load 1 bit at a time
  - Read 1 bit at a time
- Hybrid serial/parallel also possible and useful



# Shift Registers

- A shift register is a register in which
  - Binary data can be stored, and
  - This data can be shifted left or right when a shift signal is applied
- Shifts can be linear or cyclic
- Shown below: 4-bit right-shift register with serial input and output constructed from D flip-flops
  - When *Shift* = 1, clock enabled and shifting occurs on rising edge
  - When *Shift* = 0, no shifting occurs and data in register is held







# Shift Registers

- Note that shifting an unsigned binary number left or right is equivalent to multiplication or division by 2, respectively
  - Left shift  $\rightarrow \times 2$ 
    - $0011_2 = 3_{10}$
    - $0110 = 6_{10}$
    - $1100 = 12_{10}$
    - Could keep going with larger shift register
  - Right shift  $\rightarrow \div 2$ 
    - $1100_2 = 12_{10}$
    - $0110 = 6_{10}$
    - $0011 = 3_{10}$
    - $(0001.1_2 = 1.5_{10}$  but this is beyond a basic 4-bit shift register)
- Think back to multiplication of two unsigned binary numbers (Ch 1)
  - Each partial product was either
    - Zero, or
    - Multiplicand *shifted* over appropriate number of places
  - Can you envision combining shift register and adder with accumulator?

Next slide shows a little more, but...  
Multiplication in Arithmetic Logic Units (ALUs) is in higher level courses (e.g. ECE 3561, CSE 3421)



## Shift Registers

## Unsigned Binary Multiplier

1111	multiplicand	
<u>1101</u>	multiplier	← LSB = 1, keep multiplicand
1111	1st partial product	
<u>0000</u> 0	2nd partial product	← But bit = 0, add in all 0s
(01111)	sum of first two partial products	
<u>1111</u> 00	3rd partial product	← Bit = 1, Shift multiplicand left, Shift In = 0
(1001011)	sum after adding 3rd partial product	
<u>1111</u> 000	4th partial product	← Bit = 1, Shift multiplicand left, Shift In = 0
11000011	final product (sum after adding 4th partial product)	

Shift left, Shift In = 0  
11110 in register

8-bit