DIGITAL BUILDING BLOCKS

Chapter 5

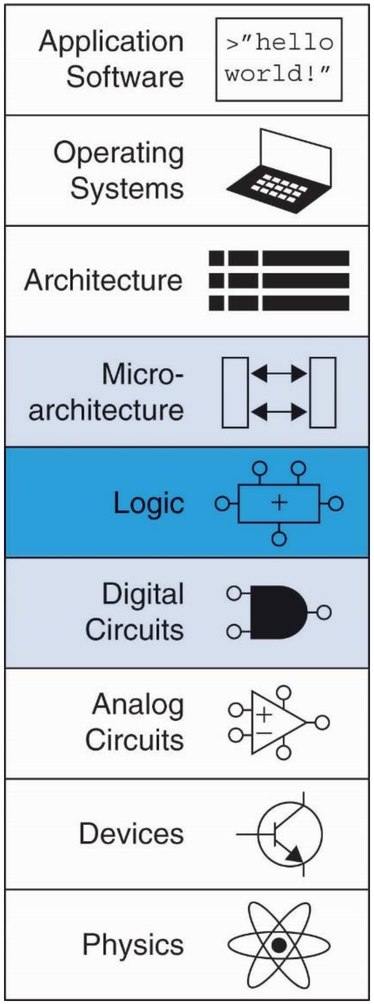
***Digital Design and Computer Architecture*, 2nd Edition**

David Money Harris and Sarah L. Harris

ELSEVIER

Chapter 5 :: Topics

- **Introduction**
- **Arithmetic Circuits**
- **Number Systems**
- **Sequential Building Blocks**
- **Memory Arrays**
- **Logic Arrays**

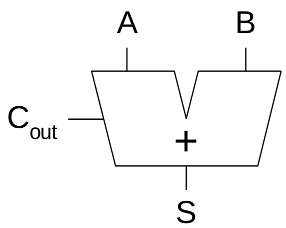


Introduction

- **Digital building blocks:**
  - Gates, multiplexers, decoders, registers, arithmetic circuits, counters, memory arrays, logic arrays
- **Building blocks demonstrate hierarchy, modularity, and regularity:**
  - Hierarchy of simpler components
  - Well-defined interfaces and functions
  - Regular structure easily extends to different sizes
- **You can use these building blocks to build a processor (see Chapter 7, CpE 300)**

Review: 1-Bit Adders

Half Adder

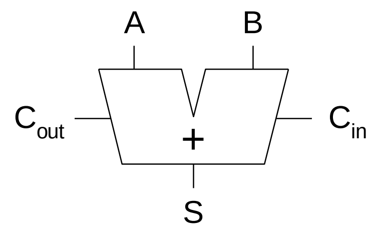


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

$S =$

$C_{out} =$

Full Adder



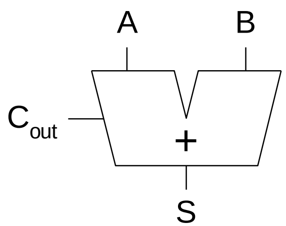
C_{in}	A	B	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

$S =$

$C_{out} =$

Review: 1-Bit Adders

Half Adder

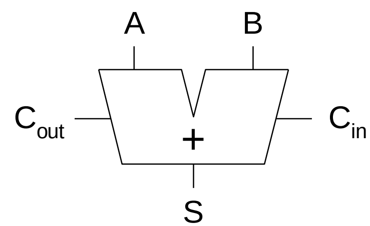


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S =$

$C_{out} =$

Full Adder



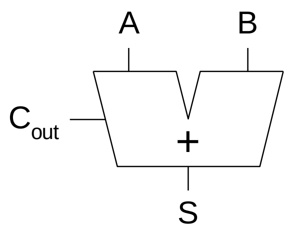
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S =$

$C_{out} =$

Review: 1-Bit Adders

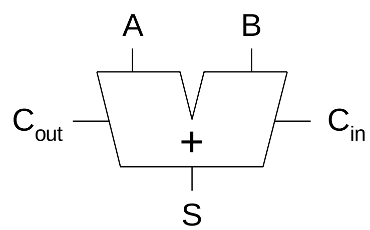
Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$

Full Adder



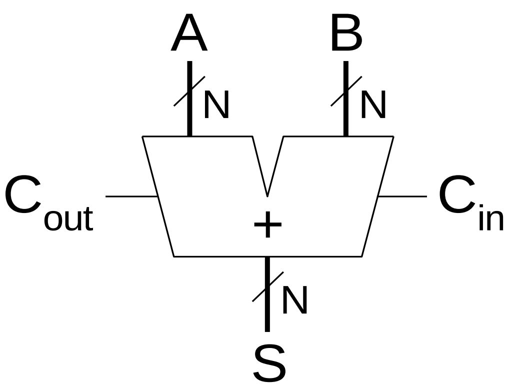
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Multibit Adders (CPAs)

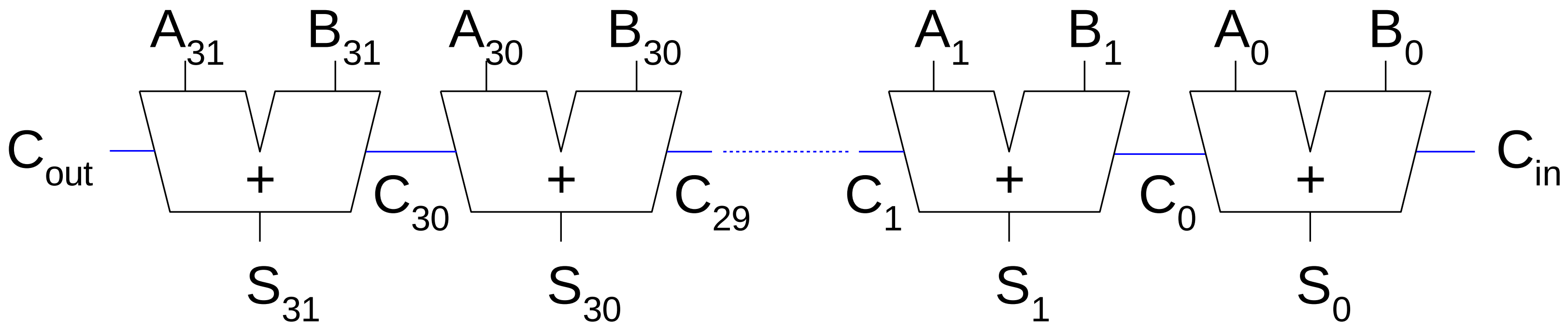
- Types of carry propagate adders (CPAs):
  - Ripple-carry (slow)
  - Carry-lookahead (fast)
  - Prefix (faster) – see book
- Carry-lookahead and prefix adders faster for large adders but require more hardware

Symbol



Ripple-Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: **slow**



Ripple-Carry Adder Delay

$$t_{\text{ripple}} = Nt_{FA}$$

where t_{FA} is the delay of a 1-bit full adder

Carry-Lookahead Adder

- **Some definitions:**
  - Column *i* produces a carry out by either **generating** a carry out or **propagating** a carry in to the carry out

Carry-Lookahead Adder

- **Some definitions:**
  - Column *i* produces a carry out by either **generating** a carry out or **propagating** a carry in to the carry out
  - Generate (G_i) and propagate (P_i) signals for each column:
    - **Generate:** Column *i* will generate a carry out if A_i AND B_i are both 1.

$$G_i = A_i B_i$$

Carry-Lookahead Adder

- **Some definitions:**
  - Column i produces a carry out by either **generating** a carry out or **propagating** a carry in to the carry out
  - Generate (G_i) and propagate (P_i) signals for each column:
    - **Generate:** Column i will generate a carry out if A_i AND B_i are both 1.

$$G_i = A_i B_i$$

    - **Propagate:** Column i will propagate a carry in to the carry out if A_i OR B_i is 1.

$$P_i = A_i + B_i$$

Carry-Lookahead Adder

- **Some definitions:**
  - Column i produces a carry out by either **generating** a carry out or **propagating** a carry in to the carry out
  - Generate (G_i) and propagate (P_i) signals for each column:
    - **Generate:** Column i will generate a carry out if A_i AND B_i are both 1.

$$G_i = A_i B_i$$

    - **Propagate:** Column i will propagate a carry in to the carry out if A_i OR B_i is 1.

$$P_i = A_i + B_i$$

    - **Carry out:** The carry out of column i (C_i) is:

$$C_i = G_i + P_i C_{i-1}$$

Carry-Lookahead Adder

- **Some definitions:**
  - Column *i* produces a carry out by either **generating** a carry out or **propagating** a carry in to the carry out
  - Generate (G_i) and propagate (P_i) signals for each column:
    - **Generate:** Column *i* will generate a carry out if A_i AND B_i are both 1.

$$G_i = A_i B_i$$

    - **Propagate:** Column *i* will propagate a carry in to the carry out if A_i OR B_i is 1.

$$P_i = A_i + B_i$$

    - **Carry out:** The carry out of column *i* (C_i) is:

$$C_i = G_i + P_i C_{i-1} = A_i B_i + (A_i + B_i) C_{i-1}$$

Carry-Lookahead Adder

Compute carry out (C_{out}) for ***k*-bit blocks** using *generate* and *propagate* signals

Carry-Lookahead Adder

- **Example:** 4-bit blocks:

Carry-Lookahead Adder

- **Example:** 4-bit blocks:

Propagate: $P_{3:0} = P_3 P_2 P_1 P_0$

- All columns must propagate

Generate: $G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0))$

- Most significant bit generates or lower bit propagates a generated carry

Carry-Lookahead Adder

- **Example:** 4-bit blocks:

Propagate: $P_{3:0} = P_3 P_2 P_1 P_0$

- All columns must propagate

Generate: $G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0))$

- Most significant bit generates or lower bit propagates a generated carry

- **Generally,**

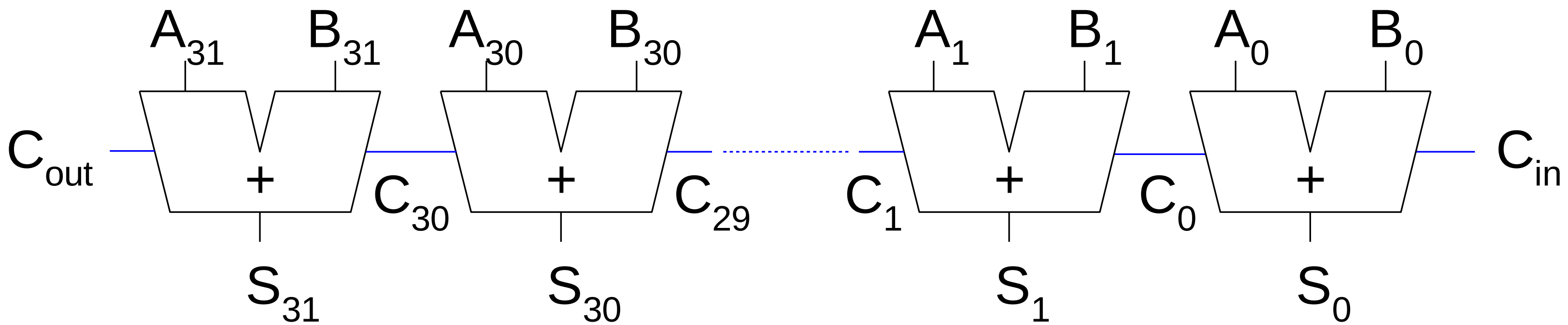
$$P_{i:j} = P_i P_{i-1} P_{i-2} P_j$$
$$G_{i:j} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} G_j)$$
$$C_i = G_{i:j} + P_{i:j} C_{j-1}$$

Carry-Lookahead Addition

- **Step 1:** Compute G_i and P_i for all columns
- **Step 2:** Compute G and P for k-bit blocks
- **Step 3:** C_{in} propagates through each k-bit propagate/generate block

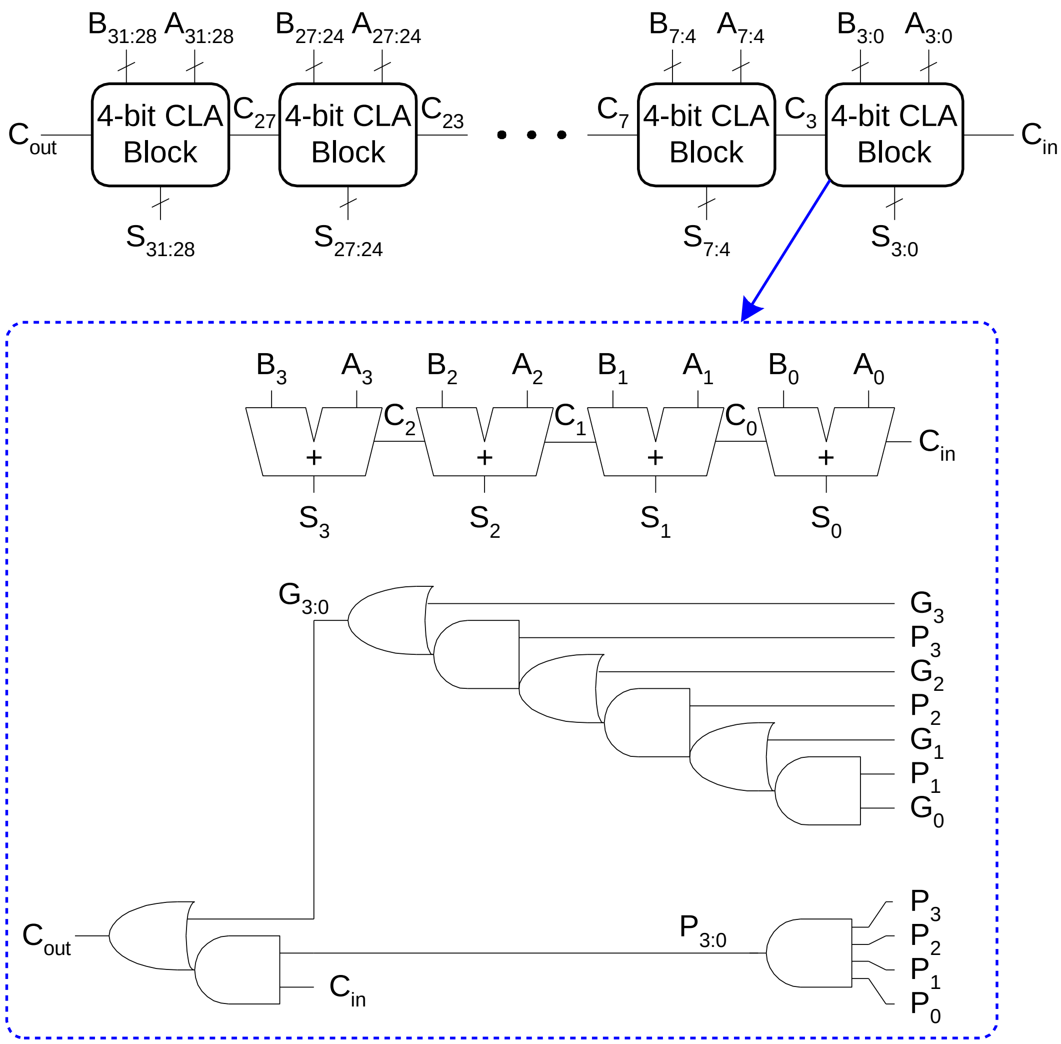
Ripple-Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: **slow**



$$t_{ripple} = Nt_{FA}$$

32-bit CLA with 4-bit Blocks



$$t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + kt_{FA}$$

Carry-Lookahead Adder Delay

For *N*-bit CLA with *k*-bit blocks:

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + kt_{FA}$$

- t_{pg} : delay to generate all P_i, G_i
- t_{pg_block} : delay to generate all $P_{i:j}$, $G_{i:j}$
- t_{AND_OR} : delay from C_{in} to C_{out} of final AND/OR gate in *k*-bit CLA block

An *N*-bit carry-lookahead adder is generally much faster than a ripple-carry adder for $N > 16$

Adder Delay Comparisons

Compare delay of 32-bit ripple-carry and carry-lookahead adders

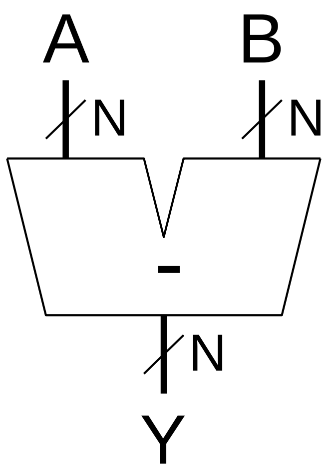
- CLA has 4-bit blocks
- 2-input gate delay = 100 ps; full adder delay = 300 ps

- Ripple
  - $t_{ripple} = Nt_{FA} = 32(300) = 9.6$ ns
- Carry-lookahead
  - $t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + k\ t_{FA}$
  - $t_{CLA} = 100 + 600 + 7(200) + 4(300) = 3.3$ ns

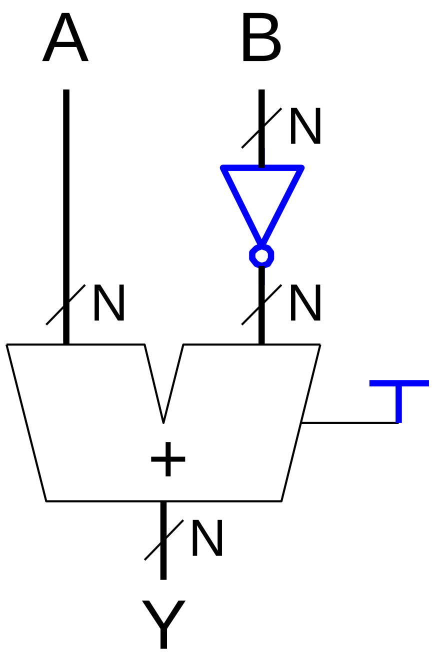
AND/OR | 6 Gates for $G_{3:0}$ | 2 Gates for $C_{in} \rightarrow C_{out}$

Subtracter

Symbol

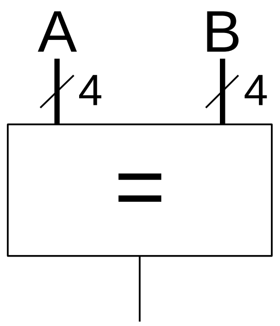


Implementation

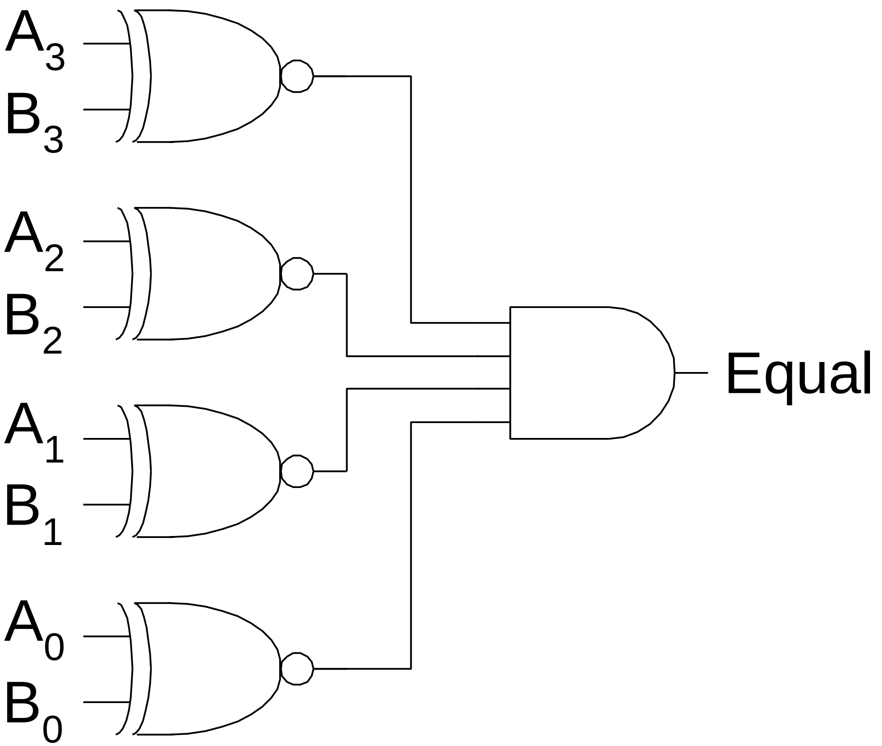


Comparator: Equality

Symbol



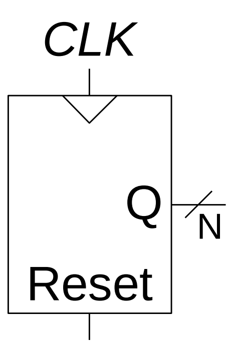
Implementation



Counters

- Increments on each clock edge
- Used to cycle through numbers. For example,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- Example uses:
  - Digital clock displays
  - Program counter: keeps track of current instruction executing

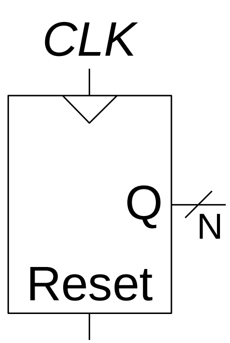
Symbol



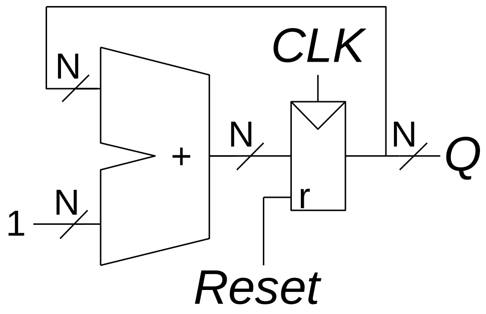
Counters

- Increments on each clock edge
- Used to cycle through numbers. For example,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- Example uses:
  - Digital clock displays
  - Program counter: keeps track of current instruction executing

Symbol



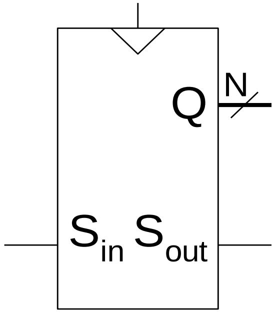
Implementation



Shift Registers

- Shift a new bit in on each clock edge
- Shift a bit out on each clock edge
- *Serial-to-parallel converter*: converts serial input (S_{in}) to parallel output ($Q_{0:N-1}$)

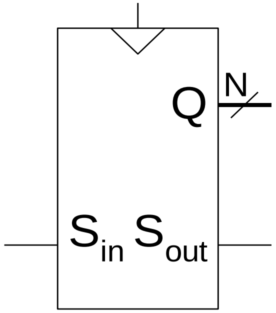
Symbol:



Shift Registers

- Shift a new bit in on each clock edge
- Shift a bit out on each clock edge
- *Serial-to-parallel converter*: converts serial input (S_{in}) to parallel output ($Q_{0:N-1}$)

Symbol:



Implementation: