Uva Wellassa University

ADDERS AND COUNTERS

Dr. Suneth Pathirana
BSc(UWU-CST)(Hons), MSc(AI)(Moratuwa), MIEEE, PhD(MSU)

Senior Lecturer
Department of Computer Science & Informatics
Uva Wellassa University of Sri Lanka.

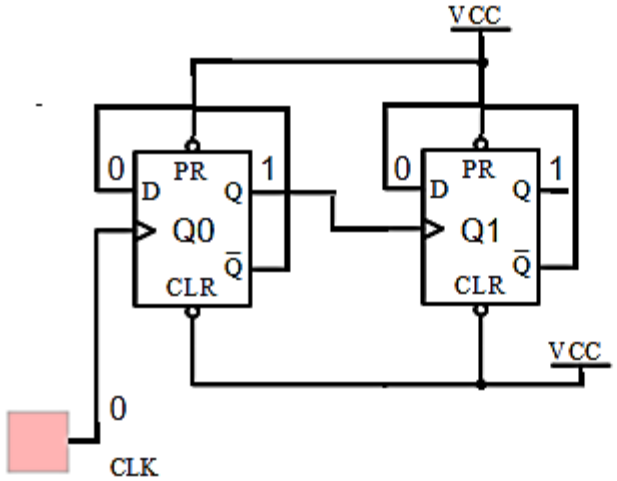
suneth@uwu.ac.lk

Counters

- A counter is a digital sequential logic device that will go through a certain predefined states (for example counting up or down) based on the application of the input pulses
- They are utilized in almost all computers and digital electronics systems
- There are two main types of counters:
  - Asynchronous and
  - Synchronous counters

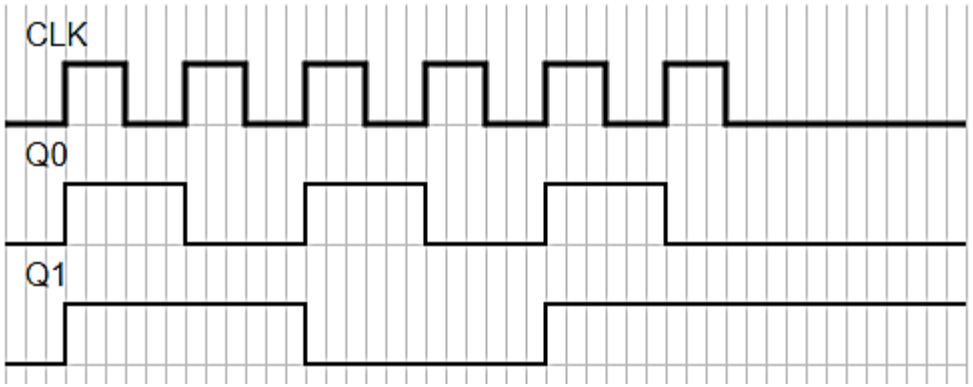
Ripple Counter

- A ripple counter is an asynchronous counter where **only the first flip-flop** is clocked by an external clock
- All subsequent flip-flops are clocked by the output of the preceding flip-flop



STATE TABLE

COUNT	Q1	Q0
3	1	1
2	1	0
1	0	1
0	0	0



Ripple Counters

- The MOD* of the ripple counter or asynchronous counter is 2^n if n flip-flops are used
- For a 4-bit counter, the range of the count is 0000 to 1111 (2^4-1)
- A counter may count up or count down or count up and down depending on the input control
- The count sequence usually repeats itself

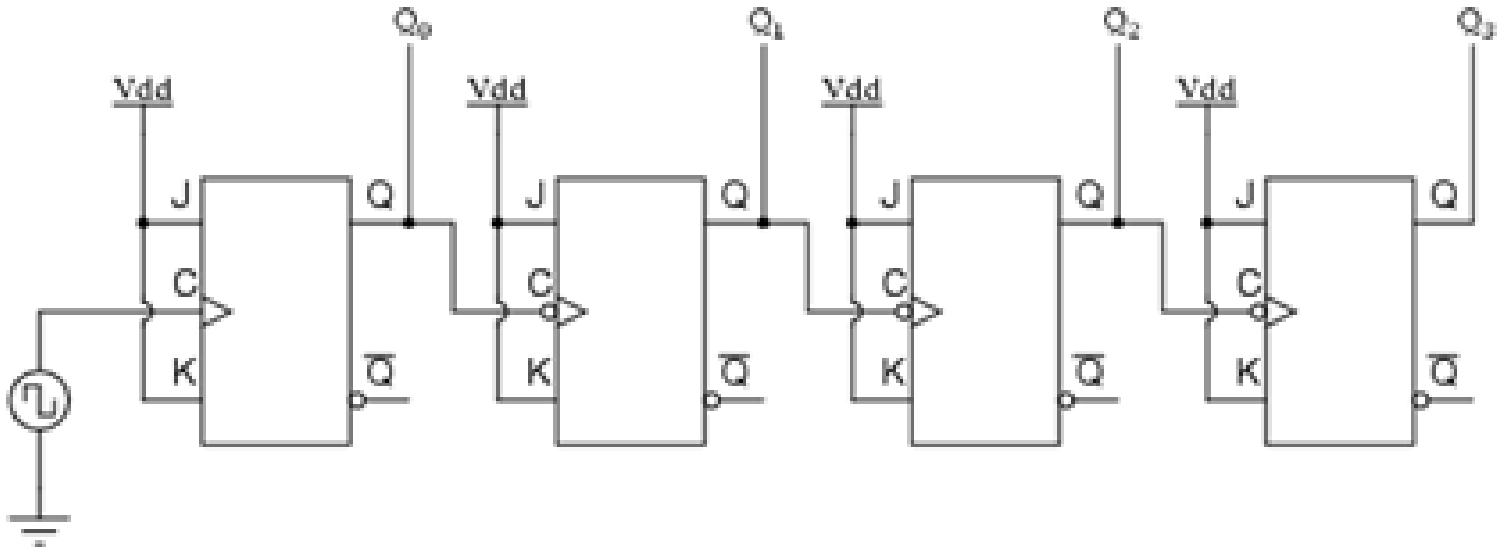
*number of unique states

Ripple Counters

- When counting up, the count sequence goes from 0000, 0001, 0010, ... 1110 , 1111 , 0000, 0001, ... etc.
  - When counting down the count sequence goes in the opposite manner: 1111, 1110, ... 0010, 0001, 0000, 1111, 1110, ... Etc.
- There are many ways to implement the ripple counter depending on the characteristics of the flip flops used and the requirements of the count sequence
  - Clock Trigger: Positive edged or Negative edged
  - JK or D flip-flops
  - Count Direction: Up, Down, or Up/Down

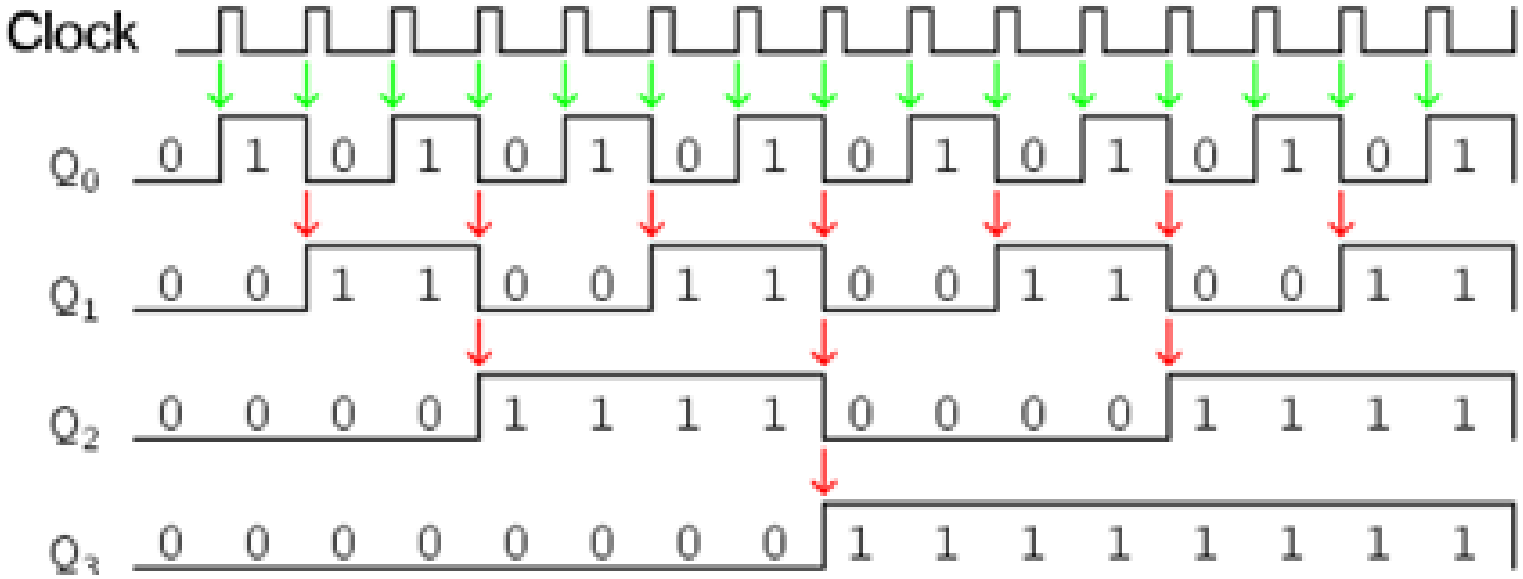
Asynchronous Counters

- This type of counters have JK Flop-Flops arranged in a way that the output of one flip-flip feeds the clock of the following flip-flop



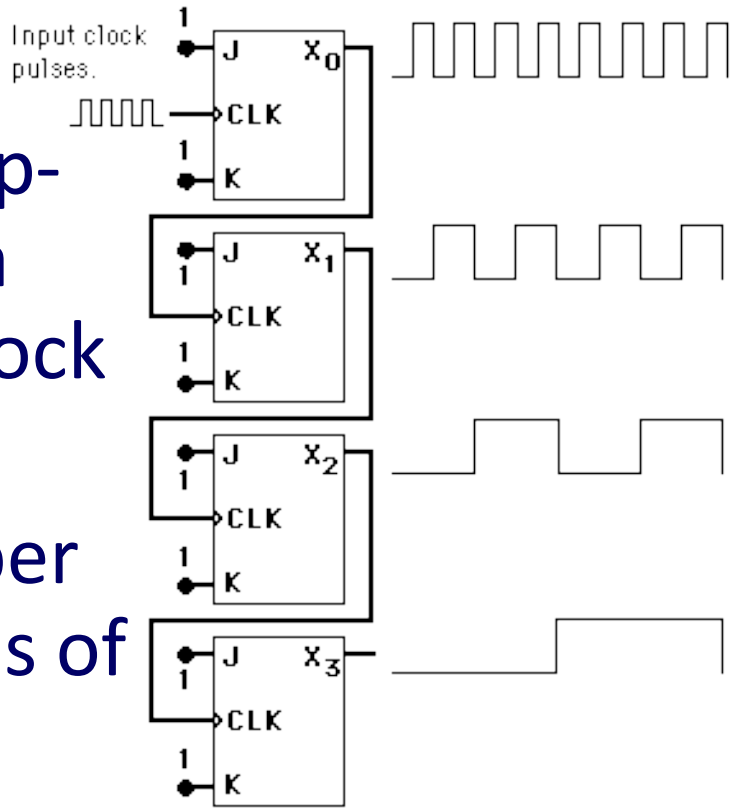
Asynchronous Counters

- The output waveforms will result from such a circuit as follows



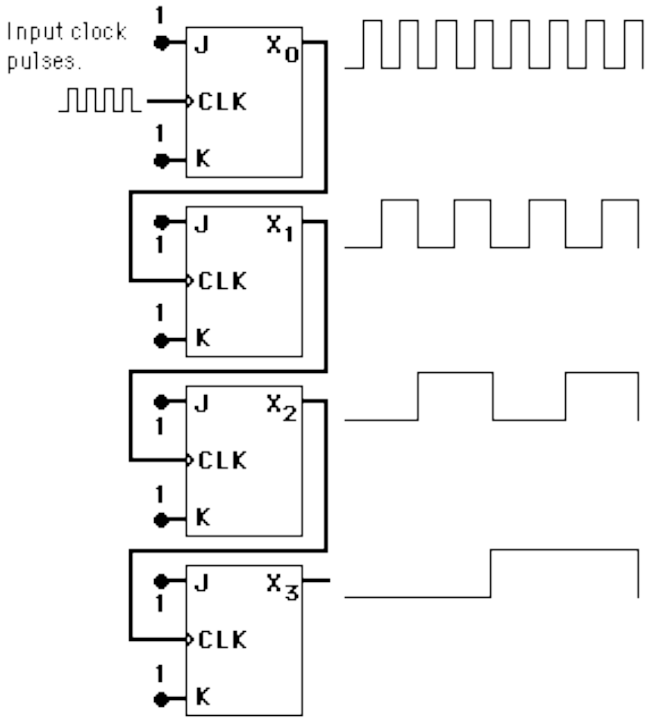
Binary Counting

- A binary counter can be constructed from J-K flip-flops by taking the output of one cell to the clock input of the next
- The J and K inputs of each flip-flop are set to 1 to produce a toggle at each cycle of the clock input
- This produces a binary number equal to the number of cycles of the input clock signal

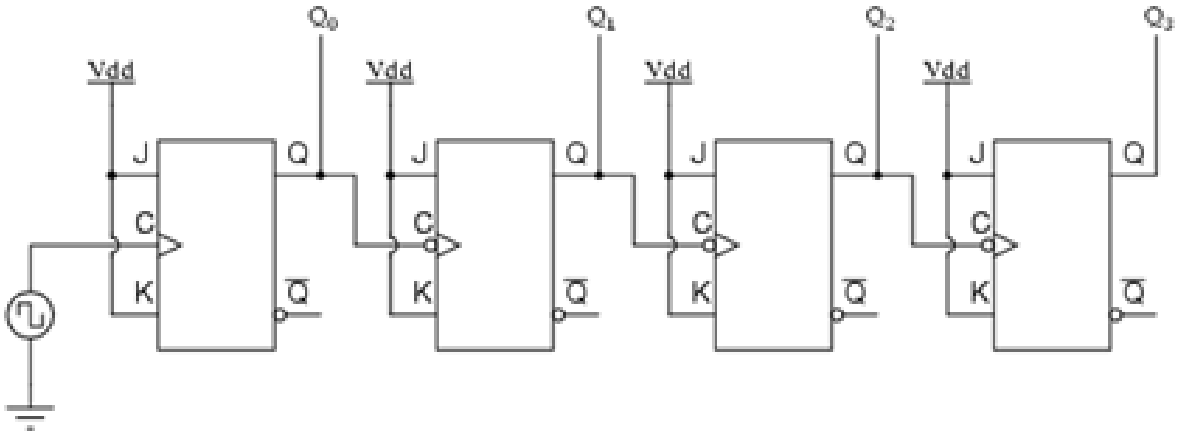


Binary Counting

- This device is sometimes called a "ripple through" counter
- The same device is useful as a frequency divider



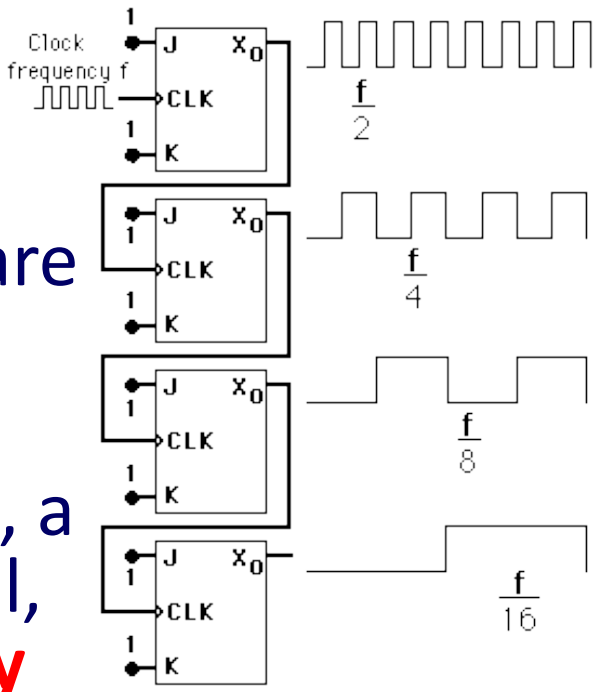
Binary Counting



CK	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

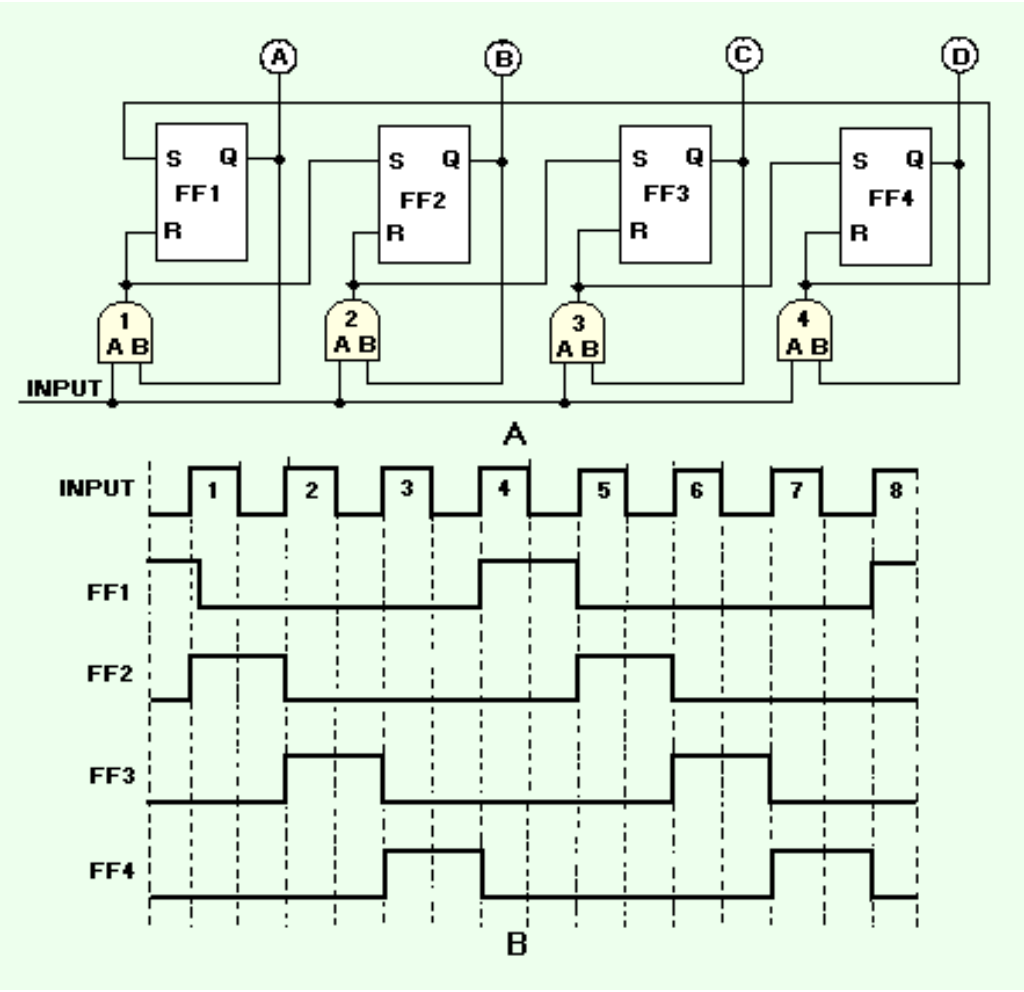
Frequency Division

- A frequency divider can be constructed from J-K flip-flops by taking the output of one cell to the clock input of the next
- The J and K inputs of each flip-flop are set to 1 to produce a toggle at each cycle of the clock input
- For each two toggles of the first cell, a toggle is produced in the second cell, so its output is at **half the frequency of the first**
- The output of the fourth cell is 1/16 the clock frequency



BCD or Decade Counter

- A decade counter is a binary counter that is designed to count to 10_{10}, or 1010_2
- An ordinary four-stage counter can be easily modified to a decade counter by adding a NAND gate

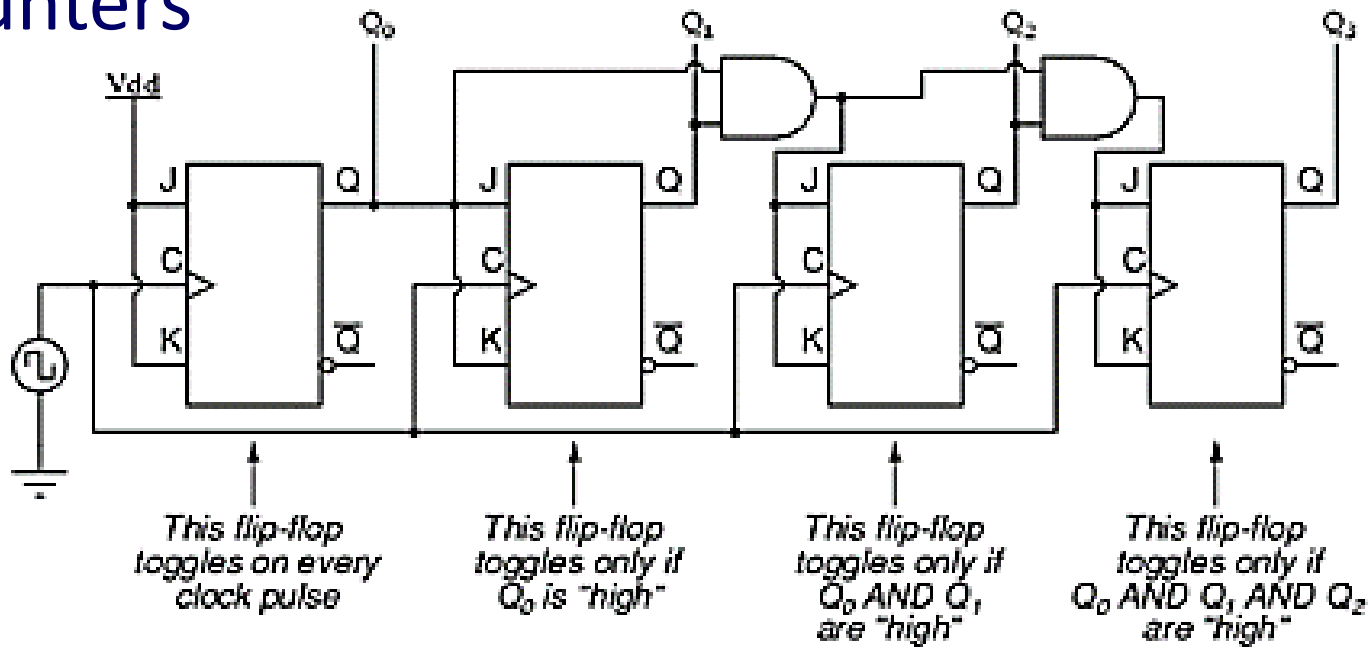


Necessity of Synchronous Counters

- Asynchronous counters are **slower than synchronous counters** because of the **delay in the transmission of the pulses** from flip-flop to flip-flop
- With a synchronous circuit, all the bits in the count change synchronously with the assertion of the clock

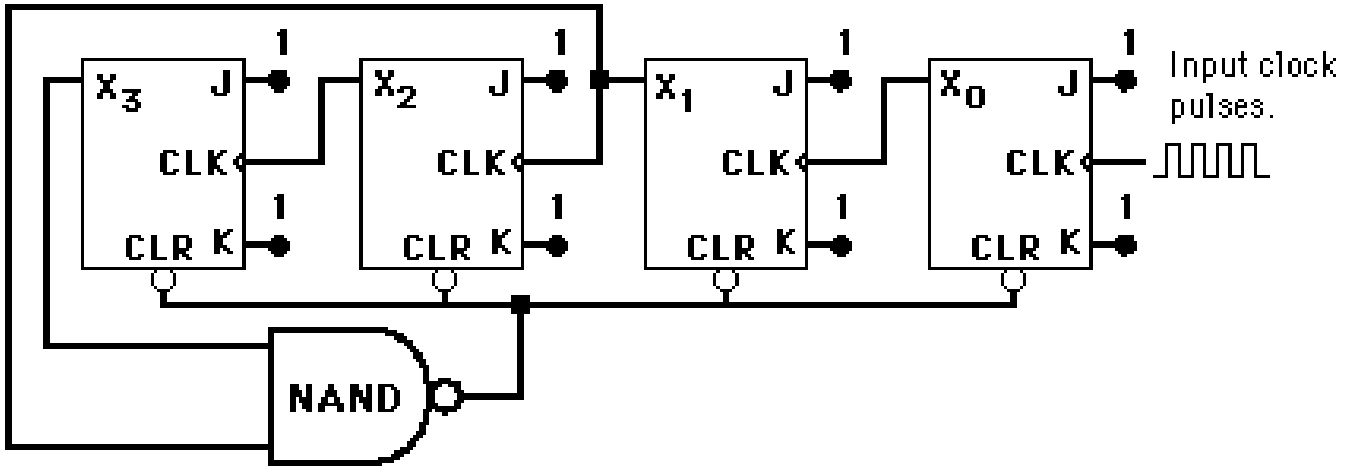
Synchronous Counters

- This type of counters has each flip-flop clocked by the same clock source, thus eliminating the cumulative delay found in asynchronous counters



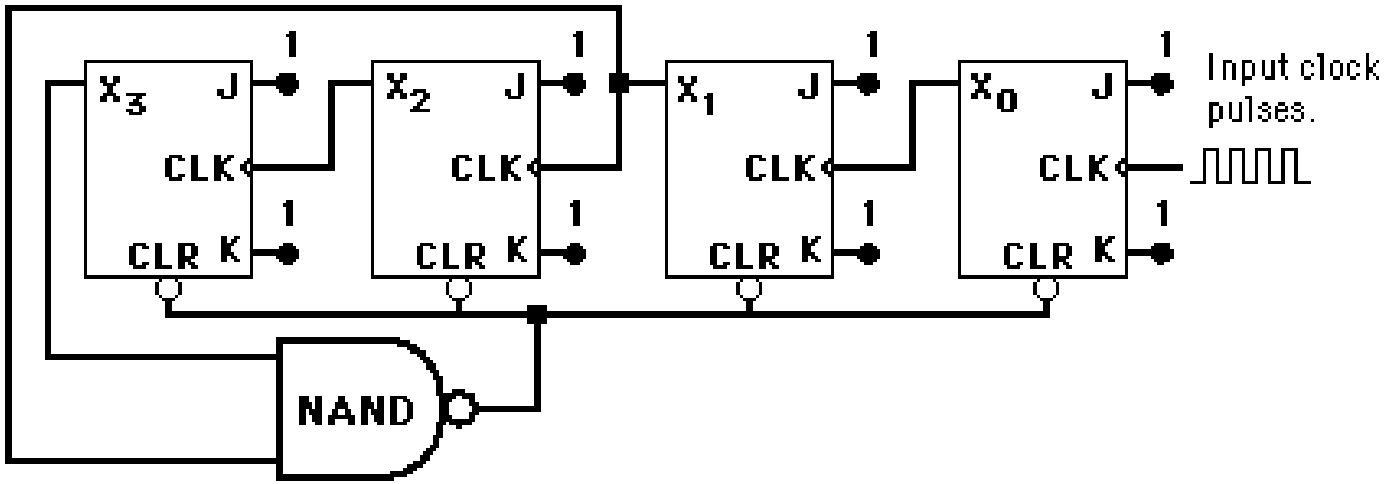
BCD or Decade Counter

- A BCD counter or decade counter can be constructed from a straight binary counter by terminating the "ripple-through" counting when the count reaches decimal 9 (binary 1001)



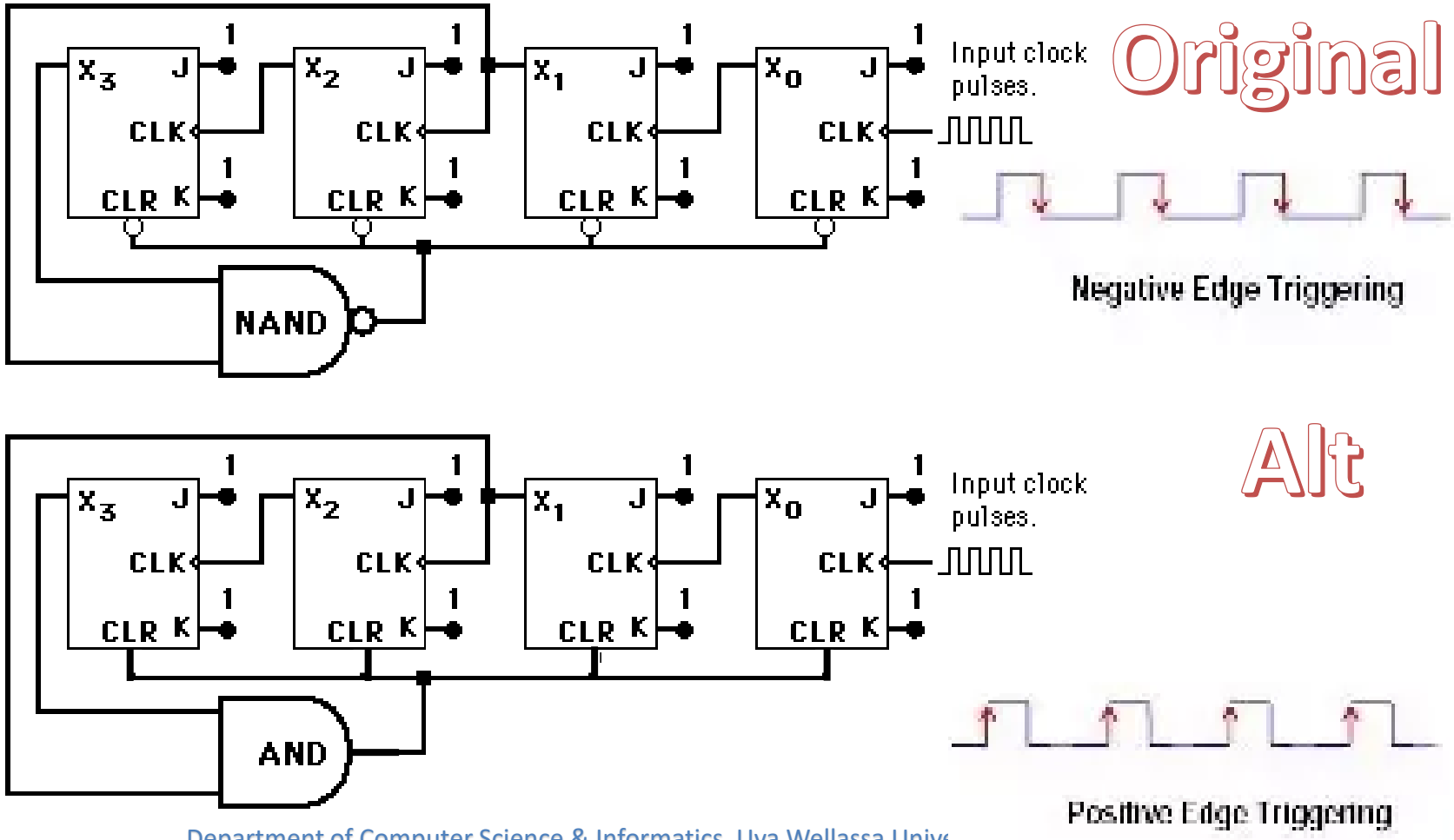
BCD or Decade Counter

- Since the next toggle would set the two most significant bits, a NAND gate tied from those two outputs to the asynchronous clear line will start the count over after 9



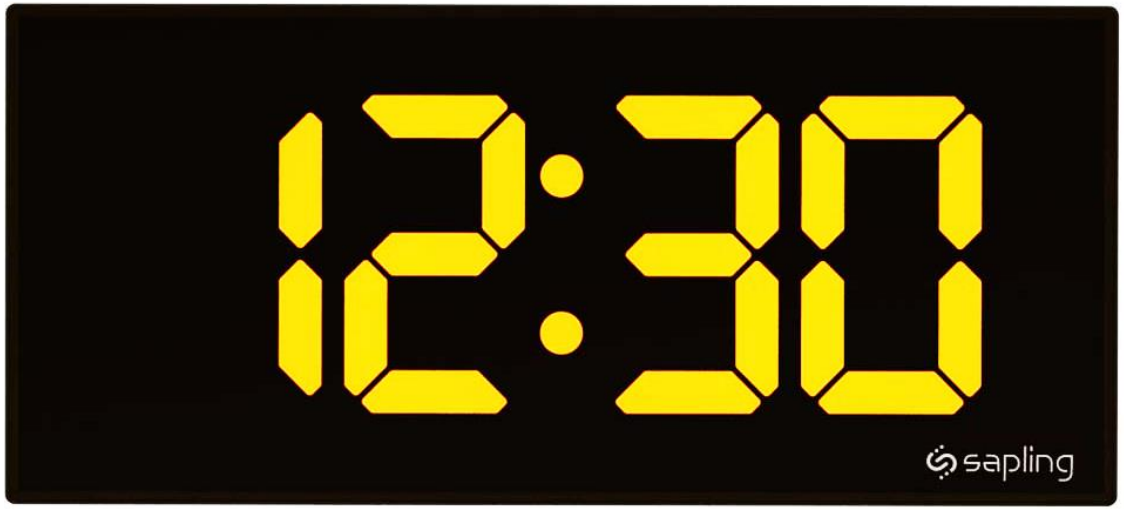
Alternative Circuit

- The below is a theoretically alternative to the previous Decade Counter
- Detect the presence of 1010_2 (= 10_{10}) then Reset



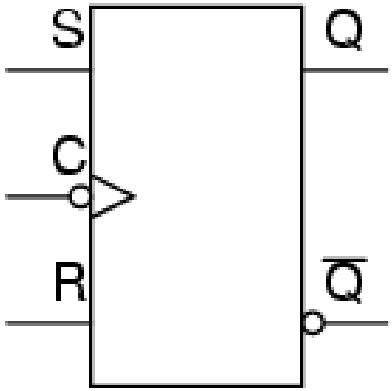
Exercise

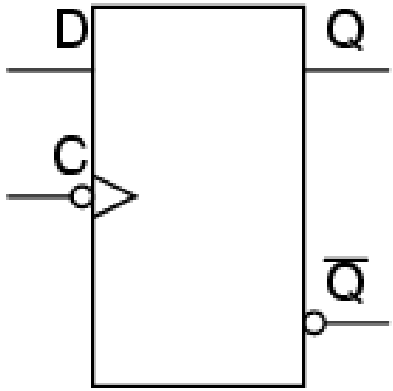
- You are supposed to design a Digital Clock (12 Hrs)
- Elaborate the strategy to be followed in order to determine the maximum value of the individual digit



Negative edge-triggered devices

- Negative edge-triggered devices are symbolized with a bubble on the clock input line

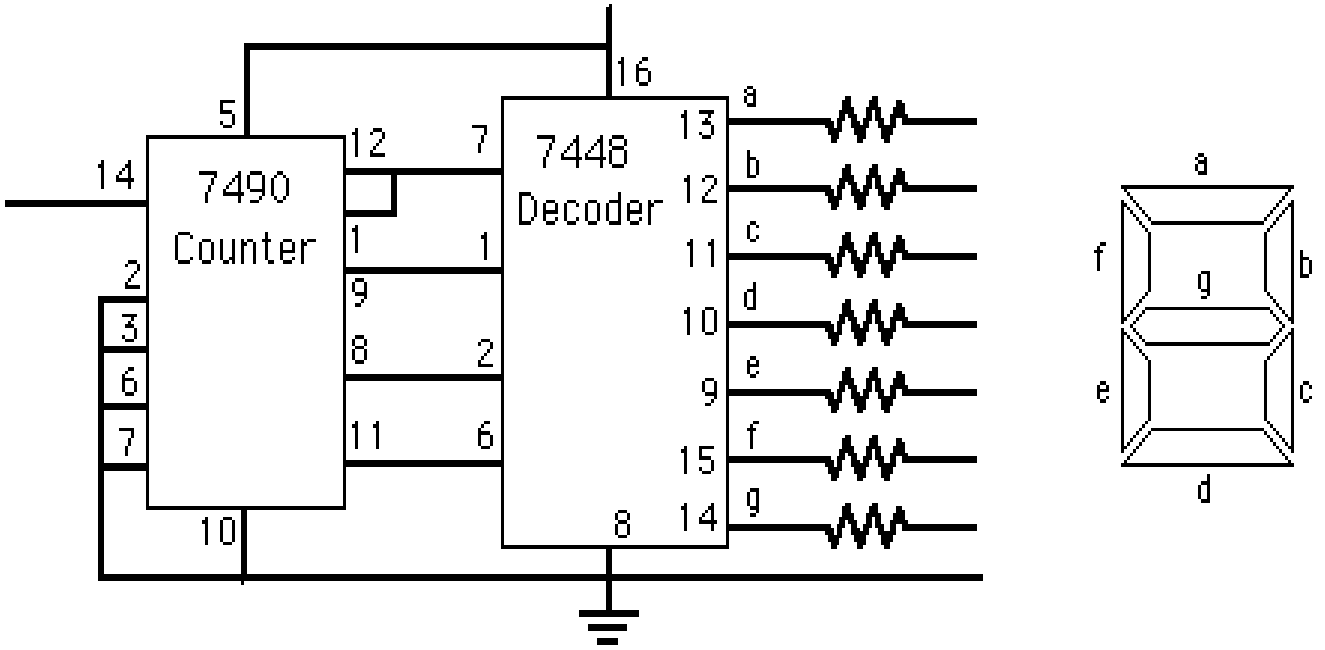




- These flip-flops will "clock" on the **falling edge** (high-to-low transition) of the clock signal
  - Note: positive edge-triggered are "clocked" on the rising edge (low-to-high transition) of the clock signal

Counter and Decoder

- The 14-pin 7490 counter chip and the 16-pin decoder chip are often used together to drive 7-segment displays



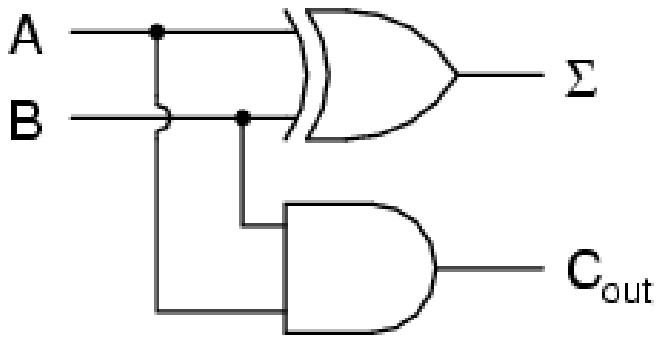
Adders

Background & Motivation

- The objective is to build a device that can add two binary digits together
- (Later, can be extended to add two binary numbers)
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - $1 + 1 = 10_2$
- So we well need two inputs (a and b) and two outputs
  - The low order output will be called Σ because it represents the sum, and
  - The high order output will be called C_{out} because it represents the carry out

Half-Adder

- So we will need two inputs (a and b) and two outputs
  - The low order output will be called Σ because it represents the sum, and
  - The high order output will be called C_{out} because it represents the carry out



Binary Addition of Two Bits

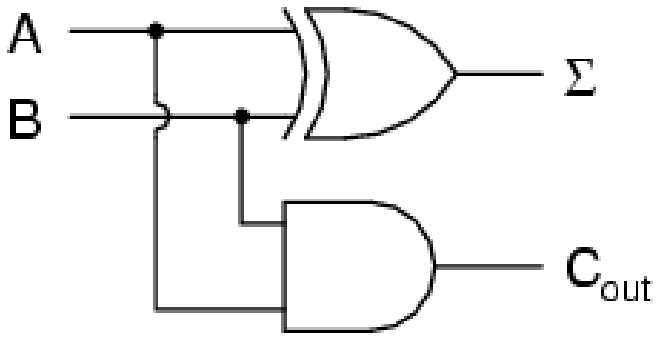
0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	(carry) 1←0

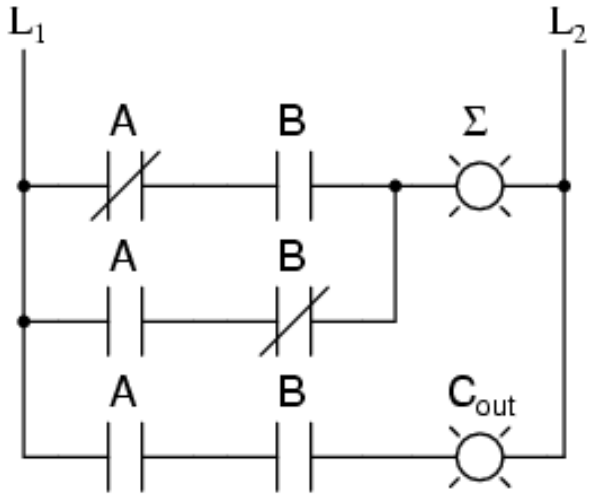
Half-Adder

- So, the truth table is…

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

- Simplifying boolean equations or making some Karnaugh map will produce the circuit shown below





Half-Adder At a Glance

- The **SUM** bit & the **CARRY** bit respectively:

$$SUM = A \text{ XOR } B = A \oplus B$$

$$CARRY = A \text{ AND } B = A.B$$

Full Adder

- This type of adder is a little more difficult to implement than a half-adder
- The main difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs
- The first two inputs are A and B and the third input is an input carry designated as C_{IN}
- When a full adder logic is designed we will be able to string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next

Why we need "Full Adder"?

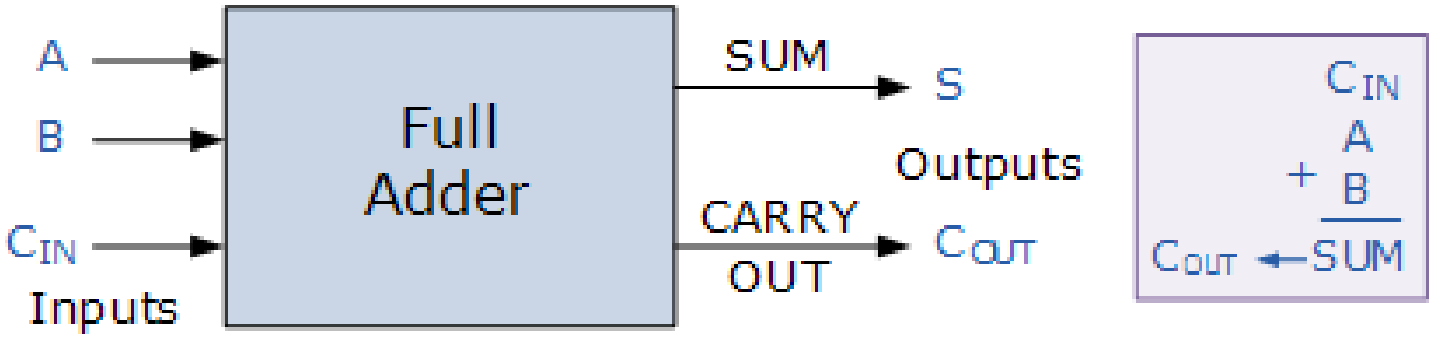
- One major disadvantage of the *Half Adder* circuit when used as a binary adder, is that there is no provision for a "Carry-in" from the previous circuit when adding together multiple data bits
- For example, suppose we want to add together two 8-bit bytes of data, any resulting carry bit would need to be able to "ripple" or move across the bit patterns starting from the least significant bit (LSB)
- The most complicated operation the half adder can do is "1 + 1" but as the half adder has no carry input the resultant added value would be incorrect
- One simple way to overcome this problem is to use a **Full Adder** type binary adder circuit

Full-Adder

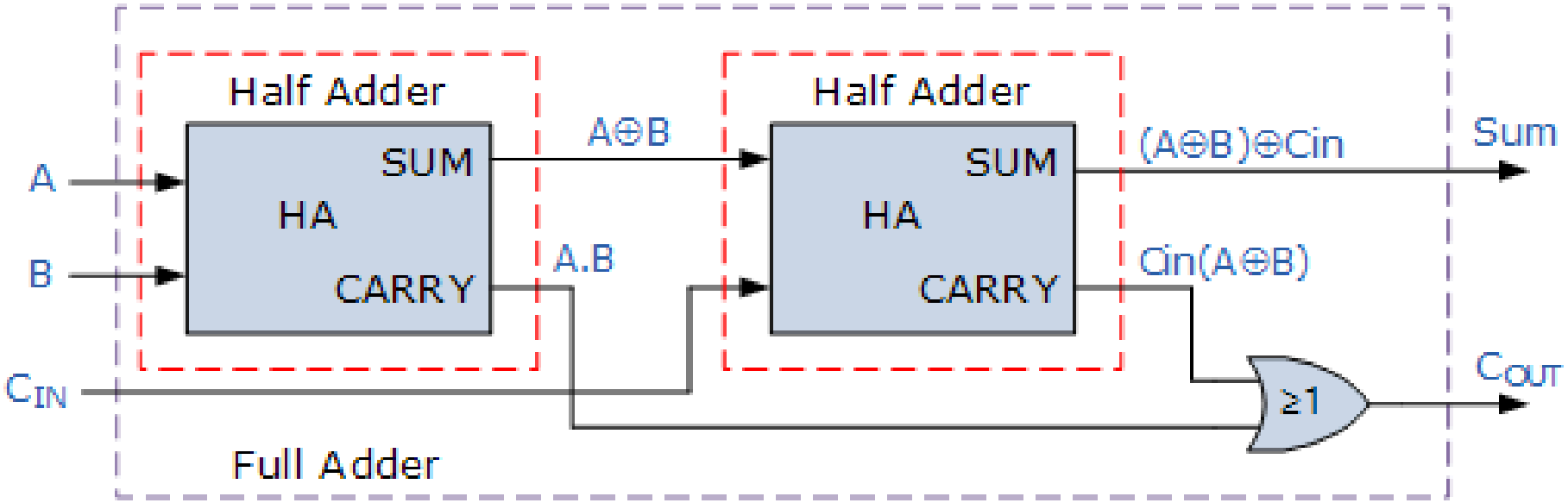
- The half-adder is extremely useful until you want to add more that one binary digit quantities
- The slow way to develop a two binary digit adders would be to make a truth table and reduce it
- Then when you decide to make a three binary digit adder, do it again
- Then when you decide to make a four digit adder, do it again

Full-Adder

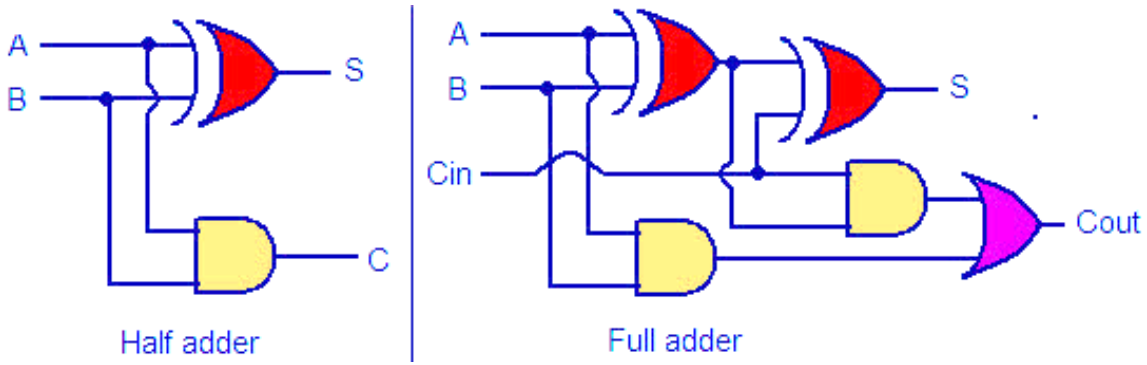
- The circuits would be fast, but development time would be slow
- Looking at a two binary digit sum shows what we need to extend addition to multiple binary digits

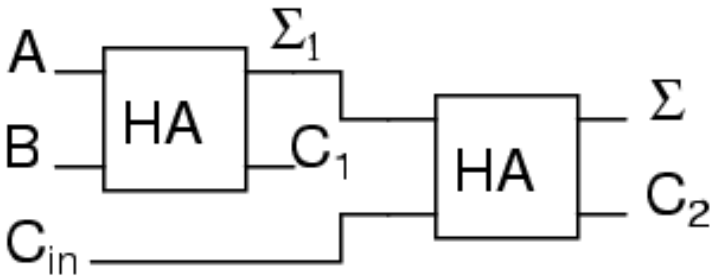


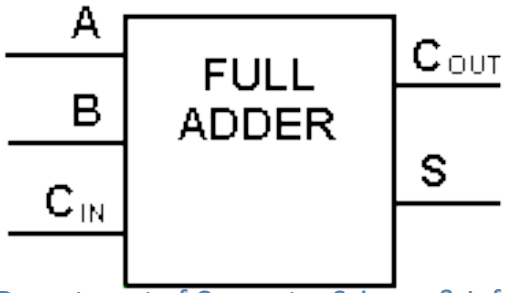
Full Adder Logic Diagram

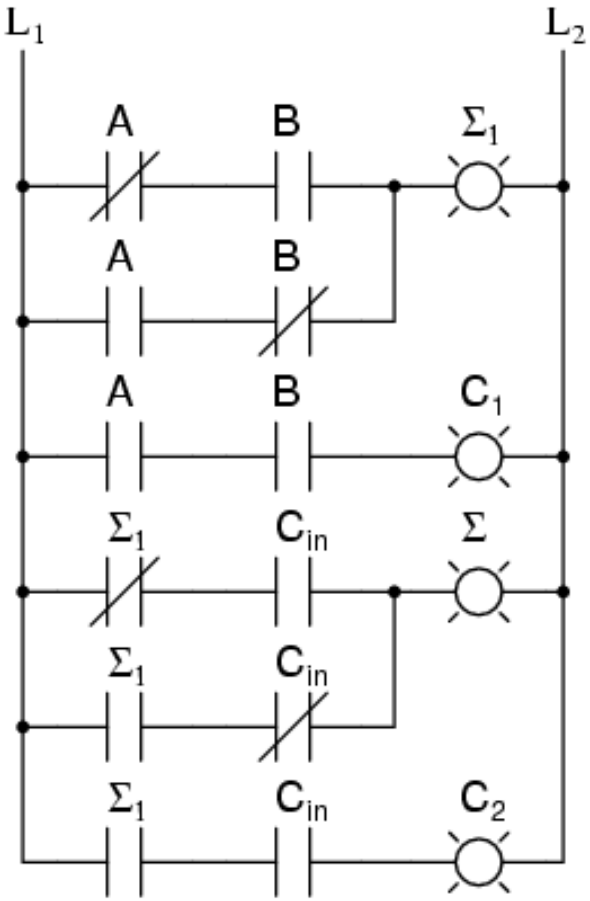


Full-Adder

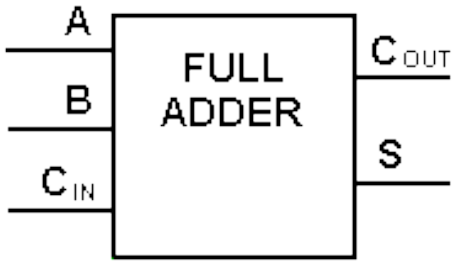

A
B
S
C
Half adder
A
B
Cin
S
Cout
Full adder


A
B
Cin
HA
Σ1
C1
HA
Σ
C2


A
B
CIN
FULL ADDER
COUT
S


L1
L2
A
B
Σ1
A
B
A
B
C1
Σ1
Cin
Σ
Σ1
Cin
Σ1
Cin
C2

Full Adder Truth Table



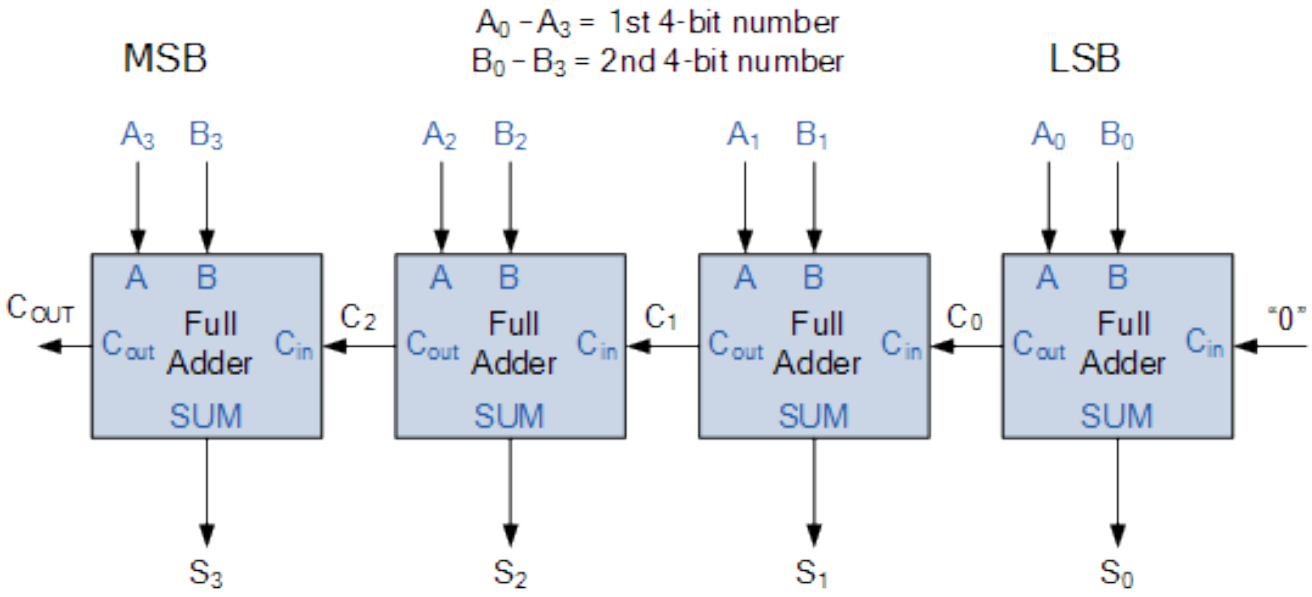
INPUTS			OUTPUT	
A	B	C-IN	C-OUT	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

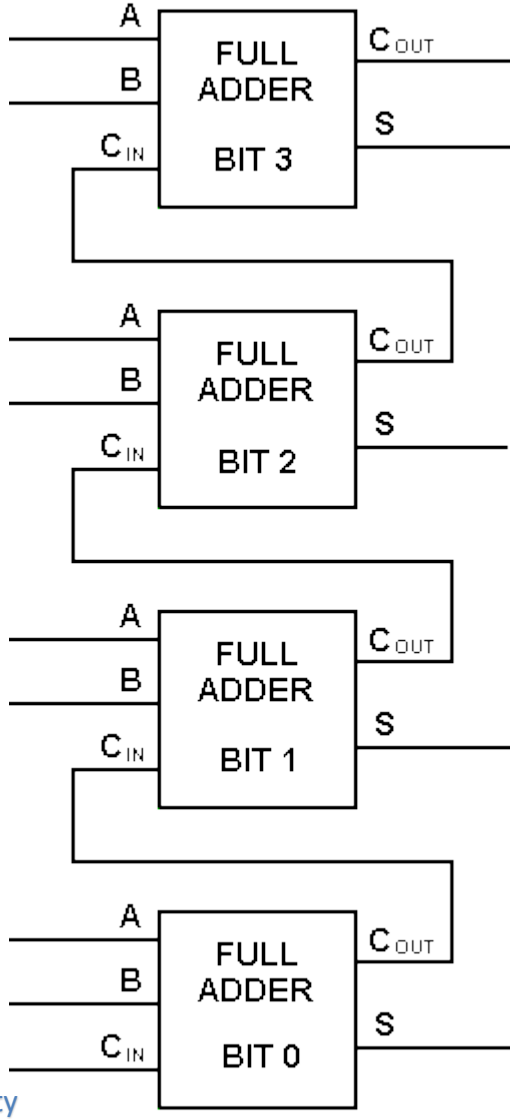
Addition of large numbers

- With this type of symbol, we can add two bits together taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude
- In a computer, for a multi-bit operation, each bit must be represented by a full adder and must be added simultaneously
- Thus, to add two 8-bit numbers, you will need 8 full adders which can be formed by cascading two of the 4-bit blocks
- The addition of two 4-bit numbers is shown in next slide

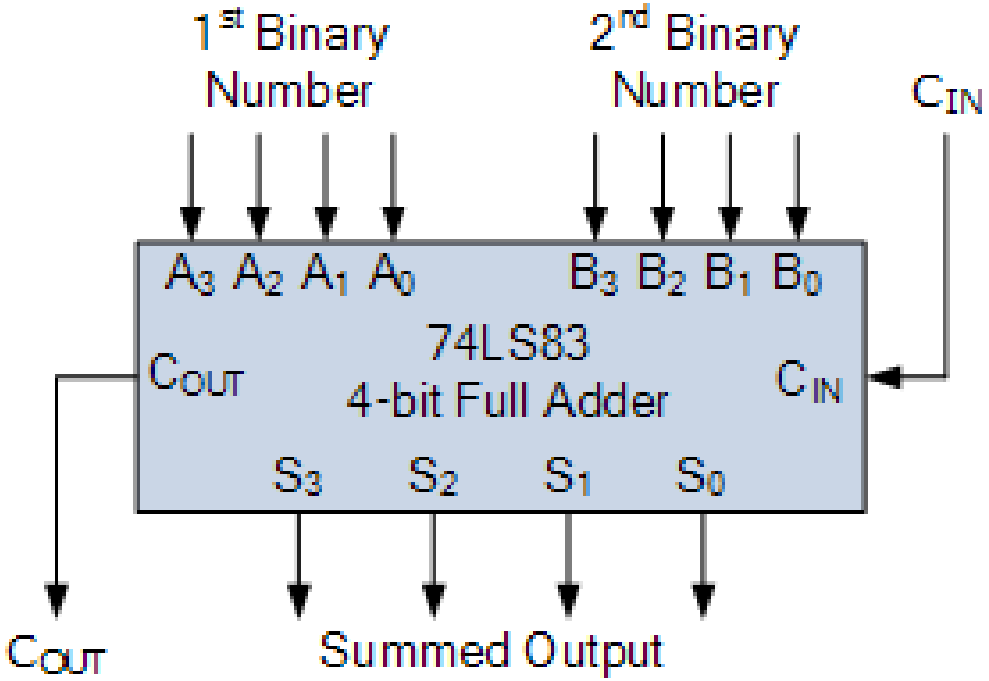
Multi-Bit Addition using Full Adder

- To add two 8-bit numbers, you will need 8 full adders which can be formed by cascading two of the 4-bit blocks





Adder ICs – 74LS83



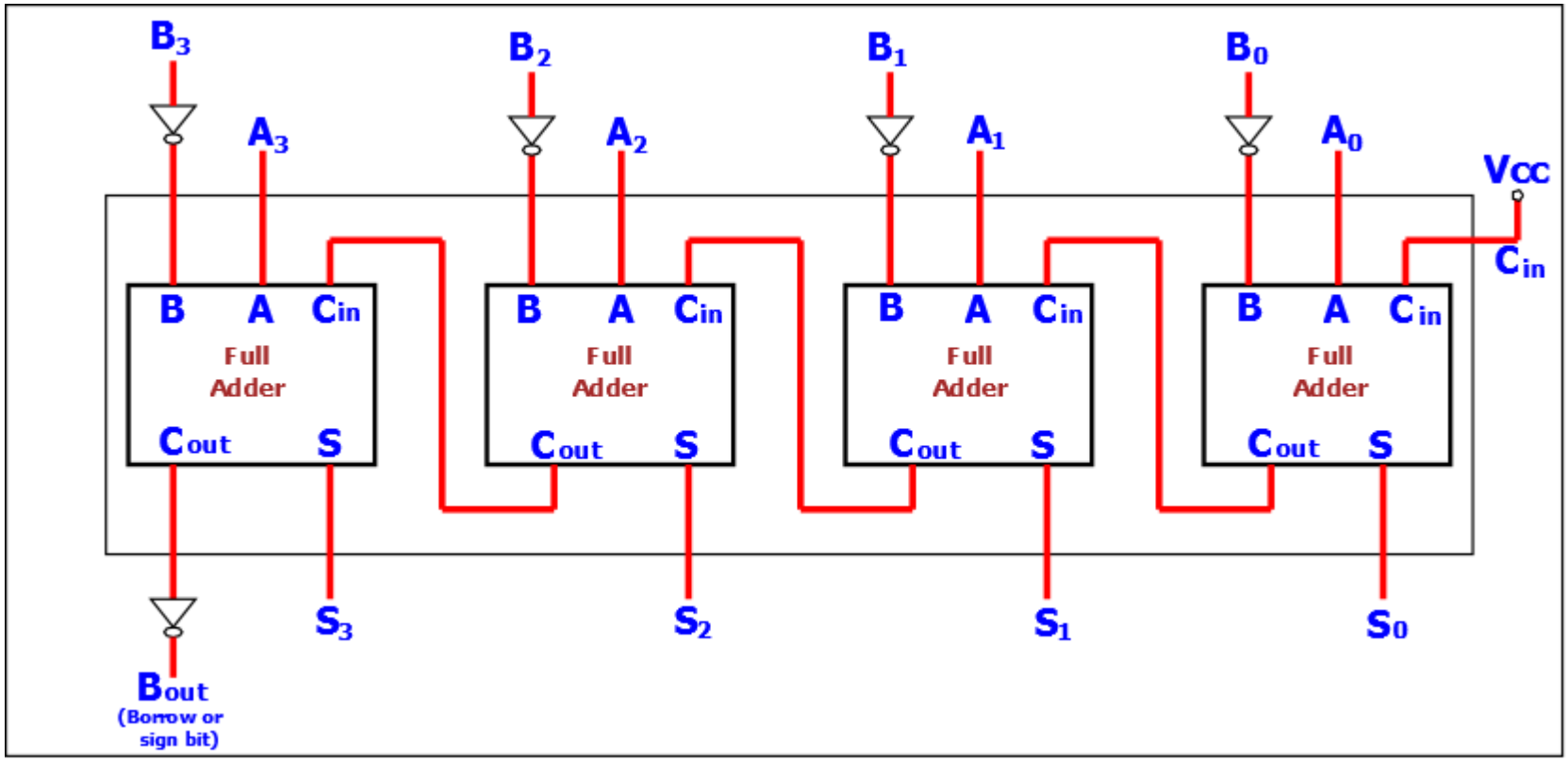
How can a full-adder be converted to Subtractor ?

- Subtraction is nothing but addition of 2's compliment
- So **A - B = A + (2's complement of B)**
  - 2' complement of B = 1's complement of B + 1
  - 1's complement of B = Inversion of B
- Example
  - A = 12, B = 7
  - 1's complement of B = 1000
  - 2's complement of B = 1001
  - So A + (2' complement of B):

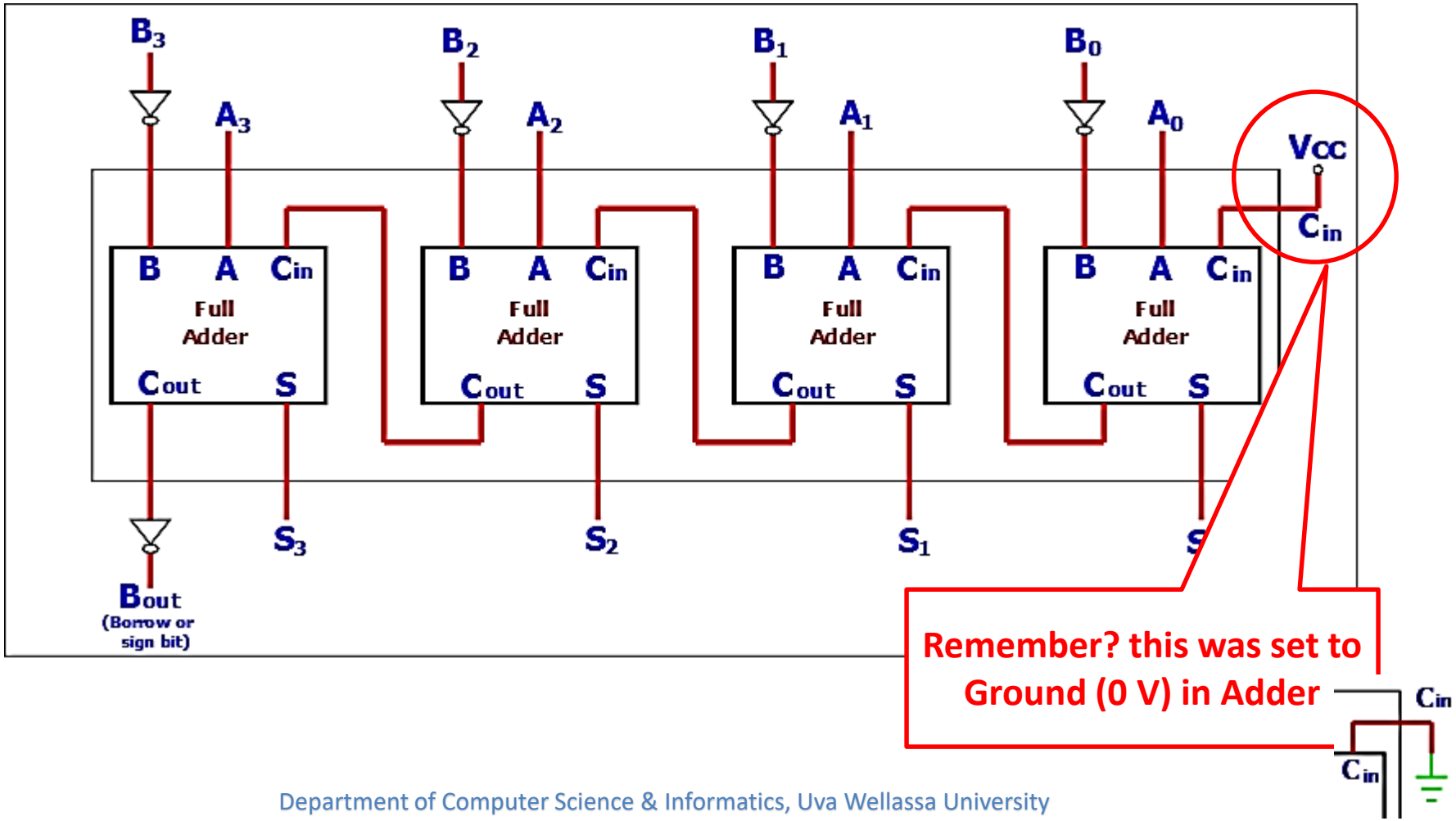
```
   1  1  0  0
 + 1  0  0  1
 ------------
 1  0  1  0  1
```

  - Inverting the Carry bit the answer will 0 0 1 0 1 = 5

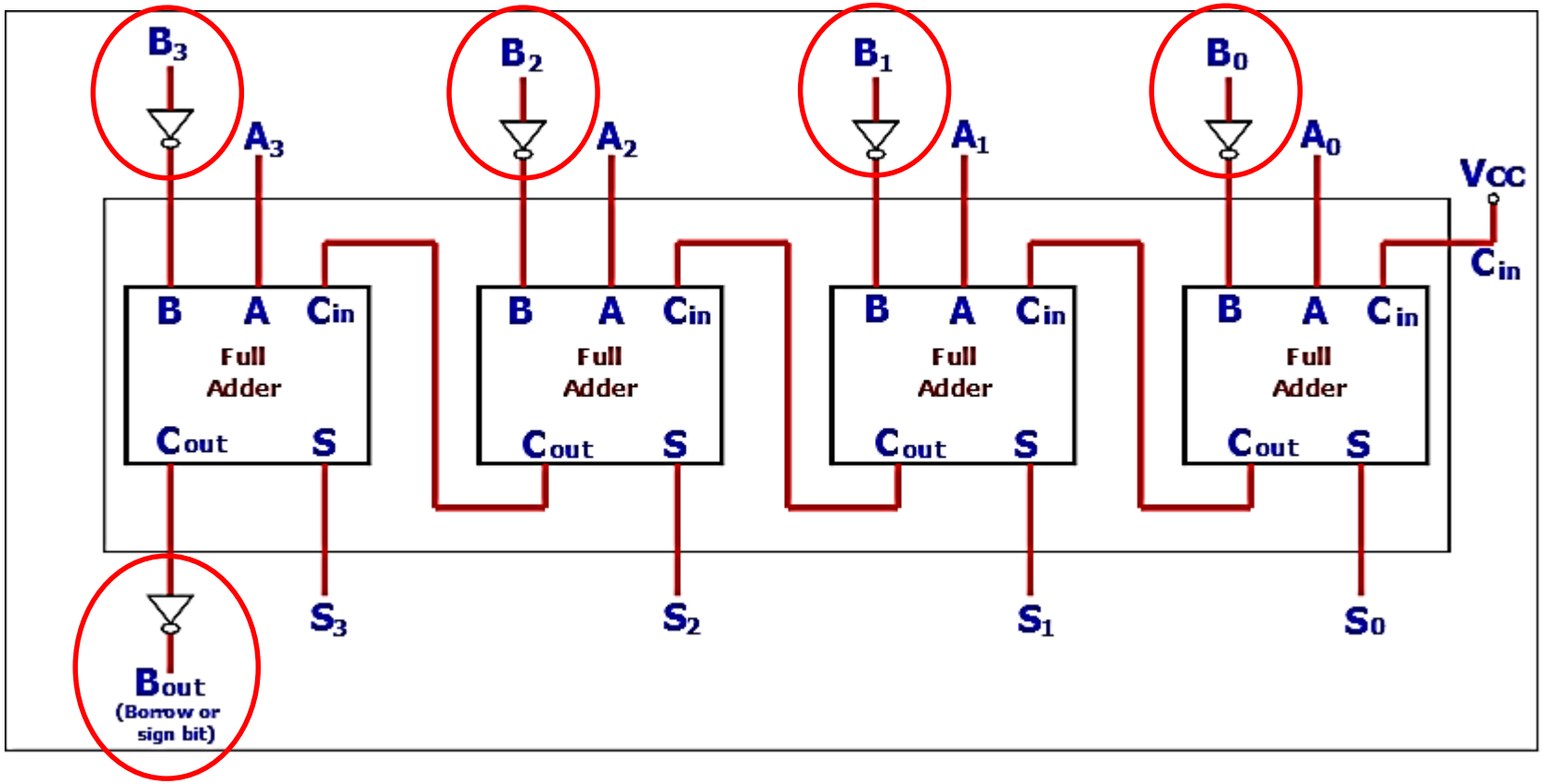
Implementation



Implementation



Implementation



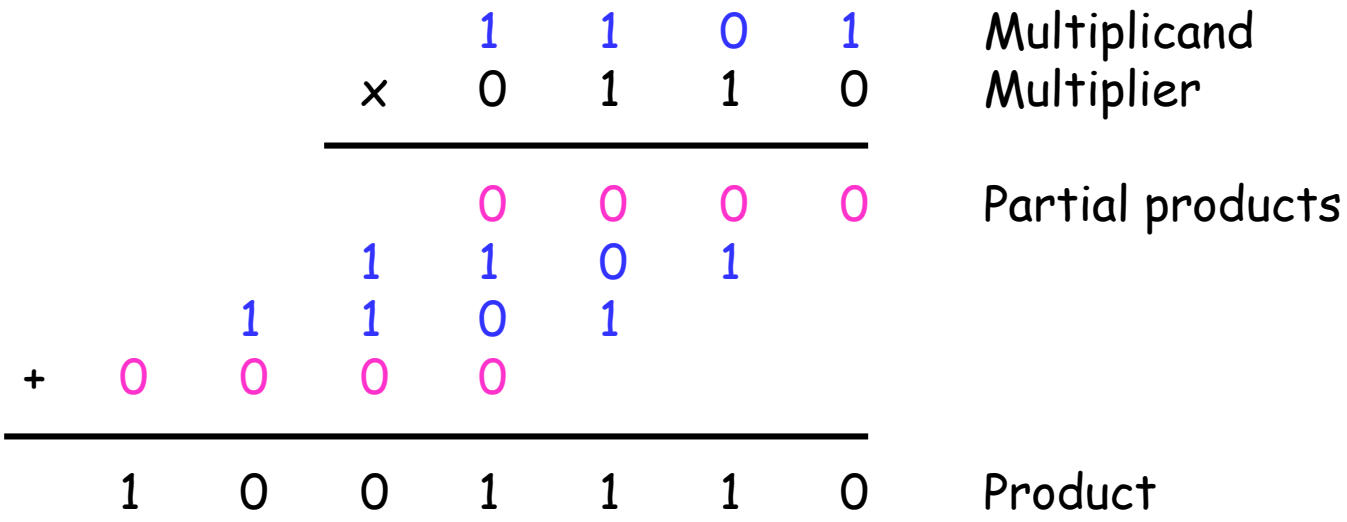
Multiplication

- Multiplication can't be that hard!
  - It's just repeated addition
  - If we have adders, we can do multiplication also
- Remember that the AND operation is equivalent to multiplication on two bits:

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a×b
0	0	0
0	1	0
1	0	0
1	1	1

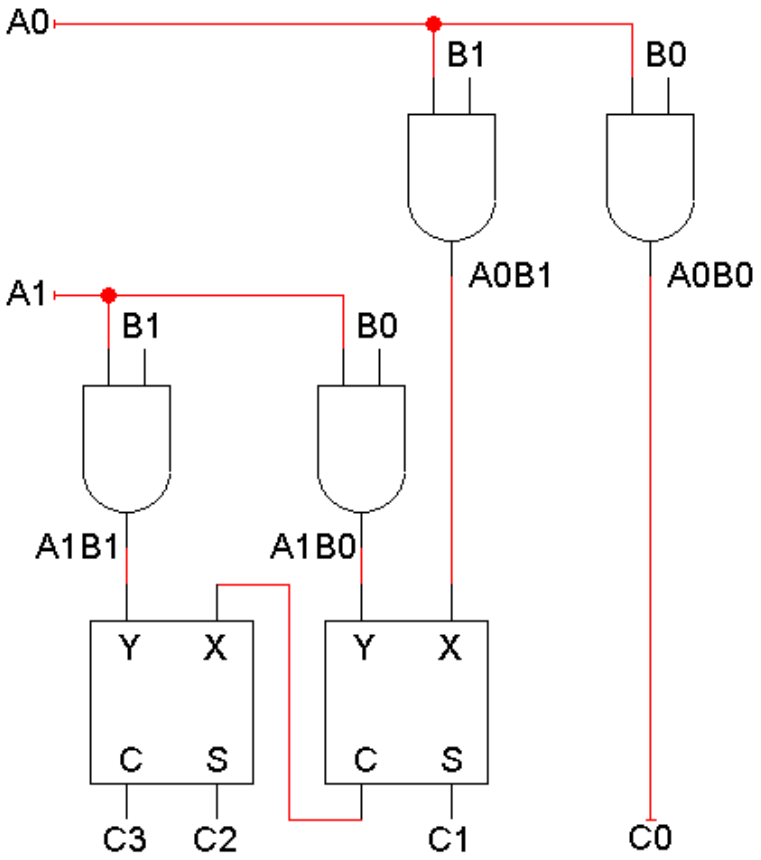
Binary multiplication example



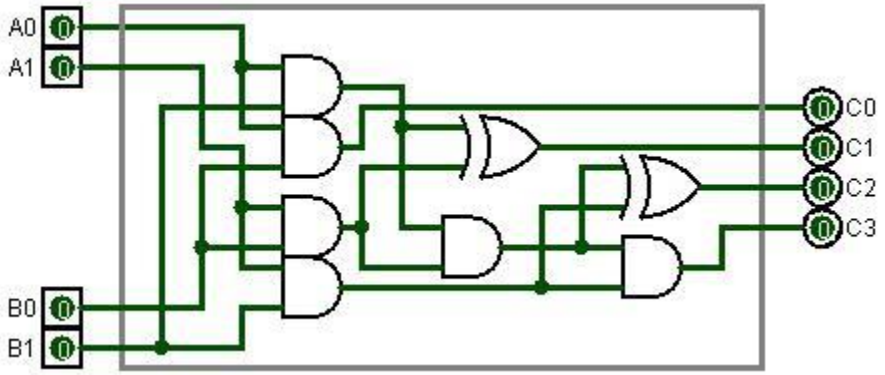
- Since we always multiply by either 0 or 1, the partial products are always either 0000 or the multiplicand (1101 in this example)
- There are four partial products which are added to form the result
  - We can add them in pairs, using three adders
  - Even though the product has up to 8 bits, we can use 4-bit adders if we "stagger" them leftwards, like the partial products themselves

A 2x2 Binary Multiplier

- The AND gates produce the partial products
- For a 2-bit by 2-bit multiplier, we can just use two half adders to sum the partial products
- In general, though, we'll need full adders
- Here C_3-C_0 are the product, not carries!



			B_1	B_0
		x	A_1	A_0
			A_0B_1	A_0B_0
+		A_1B_1	A_1B_0	
	C_3	C_2	C_1	C_0



Discussion on n-bit Binary Adders

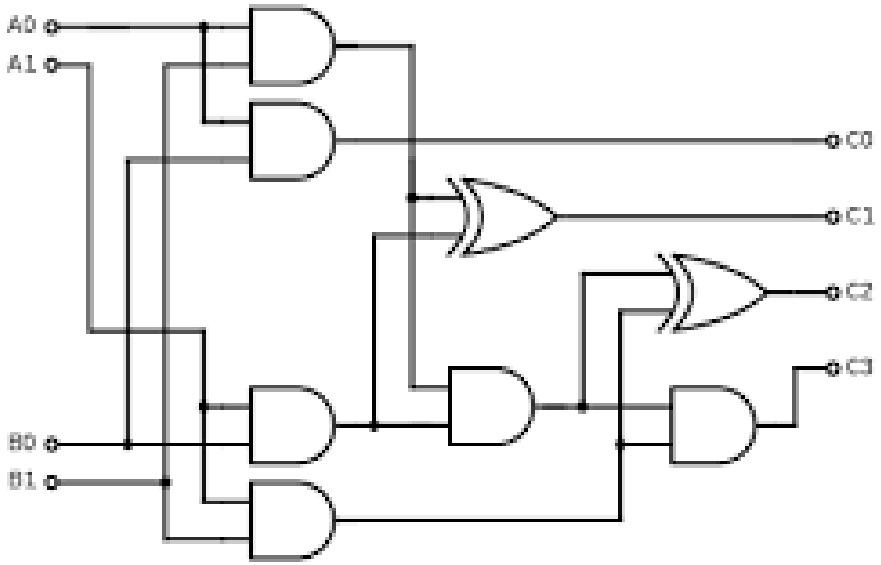
- One main disadvantage of “cascading” together 1-bit **binary adders** to add large binary numbers is that if inputs A and B change, the sum at its output will not be valid until any carry-input has “rippled” through every full adder in the chain because the MSB (most significant bit) of the sum has to wait for any changes from the carry input of the LSB (less significant bit)

Discussion on n-bit Binary Adders

- Consequently, there will be a finite delay before the output of the adder responds to any change in its inputs resulting in a accumulated delay
  - This unwanted delay time is called **Propagation delay**
- Also another problem called “overflow” occurs when an n-bit adder adds two parallel numbers together whose sum is greater than or equal to 2^n

Dividers

- Division in the context of Digital Electronics, is realized as a form of multiplication / addition
- Two Bit Binary Division



- https://www.youtube.com/watch?v=Wf_1mf6yCoc