

# **Digital Electronics II**

**Mike Brookes**

**Please pick up: Notes from the front desk**

1. What does Digital mean ?
2. Where is it used ?
3. Why is it used ?
4. What are the important features of a digital system ?

## Lecture List

- Notation, Cause and Effect
  - 1: Notation, Cause and Effect, Flipflops, Counters
- Interfacing Digital Systems
  - 2: Synchronous bit-serial Interfacing
  - 3. Asynchronous bit-serial interfacing
  - 4,5: Microprocessor-to-Memory Interface
- Synchronous State Machines
  - 6: Shift Register control and sequencing
  - 7. Data Decoding with a counter
  - 8. Synchronous state machine analysis
  - 9. Synchronous state machine design
- Digital ↔ Analog Conversion
  - 10: Digital-to-Analog conversion
  - 11. Analog-to-Digital Conversion: Flash and dither
  - 12. Analog-to-Digital Conversion: Successive approximation
- Addition Circuits
  - 13: Adders and propagation delays
  - 14. Fast Adders: bit inversion & carry lookahead
  - 15. Fast adders: Carry skip and carry save

## **Lecture Notes**

Very concise - ensure you understand each sentence.

## **Book**

Tocci, Widmer & Moss, "Digital Systems: Principles & Applications", Pearson, 11<sup>th</sup> ed, 2010.  
ISBN 0130387932

Covers most of the course though not in the same order. I do not follow any book closely.

## **Problem Sheets**

- Problems graded:
  - everyone should do A, B and C
  - D and E are harder
- Solutions are included
- Problems Class: Room 509:
  - Fri 16:00 (week 3) and Tue 3:00 (weeks 4 – 11)
- Tutorial questions

## **URL**

<http://www.ee.ic.ac.uk/hp/staff/dmb/courses/dig2/dig2.htm>

## **Discussion Group**

<http://learn.imperial.ac.uk>

## **Office Hours**

Room 812: Mon 10:00-11:00 and Fri 15:00-16:00

## Lecture 1

# Notation, Cause and Effect

### Objectives

- Introduce the IEC standard notation for logic symbols
- Emphasize the notion of cause and effect in digital circuits
- Remind you what a flipflop does
- Look at the propagation delays of a ripple counter and a synchronous counter

## Notation

### Logic Levels

A logic 1 (or high) is always the most positive of the two voltage levels.

e.g. CMOS: 0 & 5V, ECL  $-1.75$  &  $-0.9V$

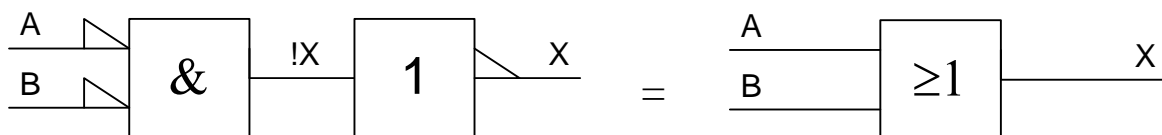
### Gates

The label indicates how many of the inputs must be high to make the output high:

- &      AND gate: all inputs high
- $\geq 1$    OR gate: one or more inputs high
- $= 1$     Exclusive-OR: exactly one input high
- $2n$     Even Parity: even number of inputs high

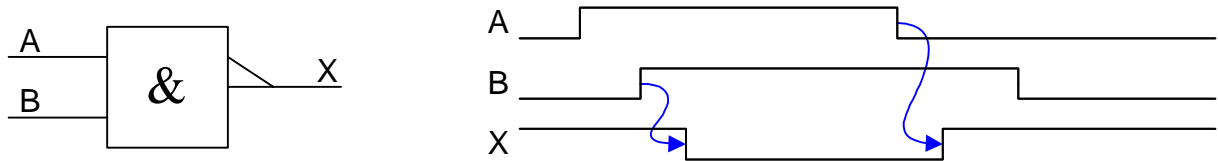
### Inversion Triangles

We can invert signals on the way in or on the way out:



$\overline{X}$  or !X denotes the inverse of X.

### Cause & Effect



Input B going high **causes** X to go low

Input A going low **causes** X to go high

#### Propagation Delay:

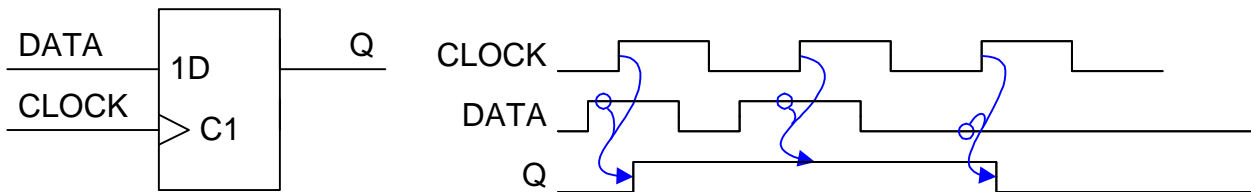
The time delay between a cause (an input changing) and its effect (an output changing).

Example: 74AC00: Advanced CMOS 2-input NAND gate

	min	typ	max	
$A\uparrow$ to $X\downarrow$ ( $t_{PHL}$ )	1.5	4.5	6.5	ns
$A\downarrow$ to $X\uparrow$ ( $t_{PLH}$ )	1.5	6.0	8.0	ns

$t_{PHL}$  and  $t_{PLH}$  refer to the direction that the output changes: high-to-low or low-to-high.

## D-Flipflop



### Notation:

- > input effect happens on the rising edge
- C1 C  $\Rightarrow$  Clock input, 1  $\Rightarrow$  This input is input number 1.
- 1D D  $\Rightarrow$  Data input,  
1  $\Rightarrow$  This input is controlled by input number 1.

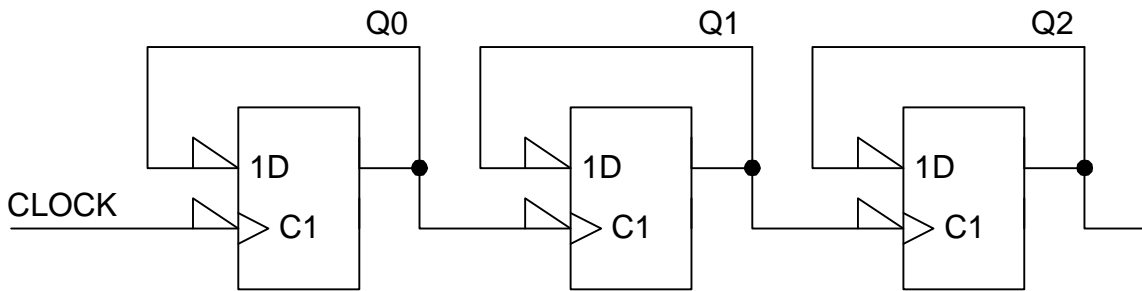
The meaning of a number depends on its position:

A number after a letter is used to identify a particular input.  
A number before a letter means that this input is controlled by one of the other inputs.

### Cause and Effect:

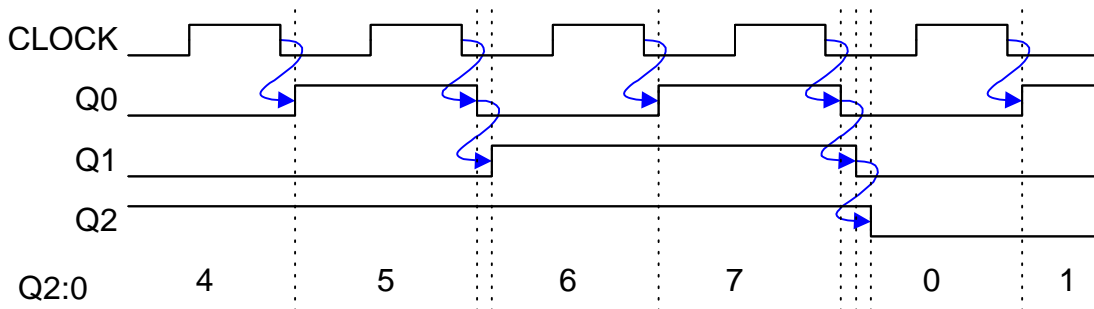
- CLOCK $\uparrow$  causes Q to change after a short delay. This is the only time Q ever changes.
- The value of D just before CLOCK $\uparrow$  is the new Q.
- Propagation delay CLOCK $\uparrow$  to Q is typically 6 ns.
- Propagation delay DATA to Q **does not make sense** since DATA changing does not cause Q to change.

## Ripple Counter

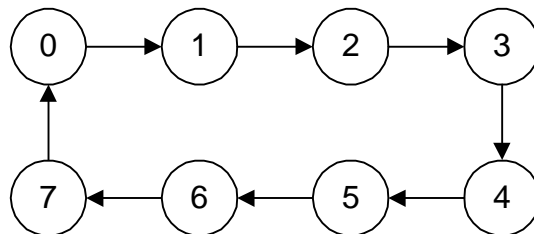


### Notation:

- Notice inverters on the CLOCK and DATA inputs
- Least significant bit of a number is always labelled 0



### State Diagram (not including transient states):



Propagation Delay: CLOCK ↓ to Q2 = 3 × 6 ns = 18 ns





## Dependency Notation

### Input Labels:

Inputs are labelled with a function letter to show what effect they have on the circuit. They have this effect whenever they are high (i.e. at logic 1).

The function letter is usually followed by an identification number (which must be unique):

- C1      Clock number 1
- M7      Mode input number 7
- D      Data input (no identification number)

### Dependencies:

If an input is affected by one or more other signals, we list their identification numbers *in front* of the function letter:

- 3,2,5D      Data input affected by input 3,2 and 5 in that order.

The identification number is used to show which of the other inputs are affected by putting it *in front* of their function letters (if any).

### Device Types:

The overall function of a device is indicated at the top of its symbol. Anything unlabelled is a flipflop or register.

## Function Letters for Input Signals

A	Address inputs for a memory circuit
CI,CO	Carry In and Out for an adder
C	Clock or Control input
CT=xx	Set contents of register or counter to xx
D	Data input to flipflop
EN	Enable tri-state outputs
G	“Gating” input: allows signals through when high
J,K,T	Inputs for JK and Toggle flipflops
M	Mode input: selects one of several operating modes (e.g. count up or count down)
P,Q	Input numbers for adders, multipliers etc.
R,S	Reset and Set inputs
V	Forces a signal to 1 when high
+, -	Increment or Decrement
←, →	Shift up (left) or shift down (right)

## Device Types

&, ≥1, =1	Gates
(blank)	Latch, Flipflop or register
MUX	Multiplexer
Σ	Adder
Π	Multiplier
CTR	Counter
SRG	Shift Register
RAM	Read/Write memory

Note: These lists are for reference only. You are not expected to memorize them.

## Quiz Questions

1. The voltage levels for the TTL logic family are 0.4 V and 2.8 V. Which one of these corresponds to logic 1?
2. If a gate is labelled  $\geq 1$ , under what circumstances will the output be high?
3. What does the *propagation delay* of a circuit mean?
4. Why does it make no sense to talk about the propagation delay between a flipflop's DATA input and the flipflop's output?
5. A flipflop's inputs are labelled C1 and 1D respectively. Why does the 1 come after the C but before the D?
6. What is the meaning of the > sign just before the C1 in a flipflop's symbol?
7. What is the meaning of a triangle drawn where an input or output wire meets a logic symbol?
8. What is a *register*?

Answers are all in the notes.

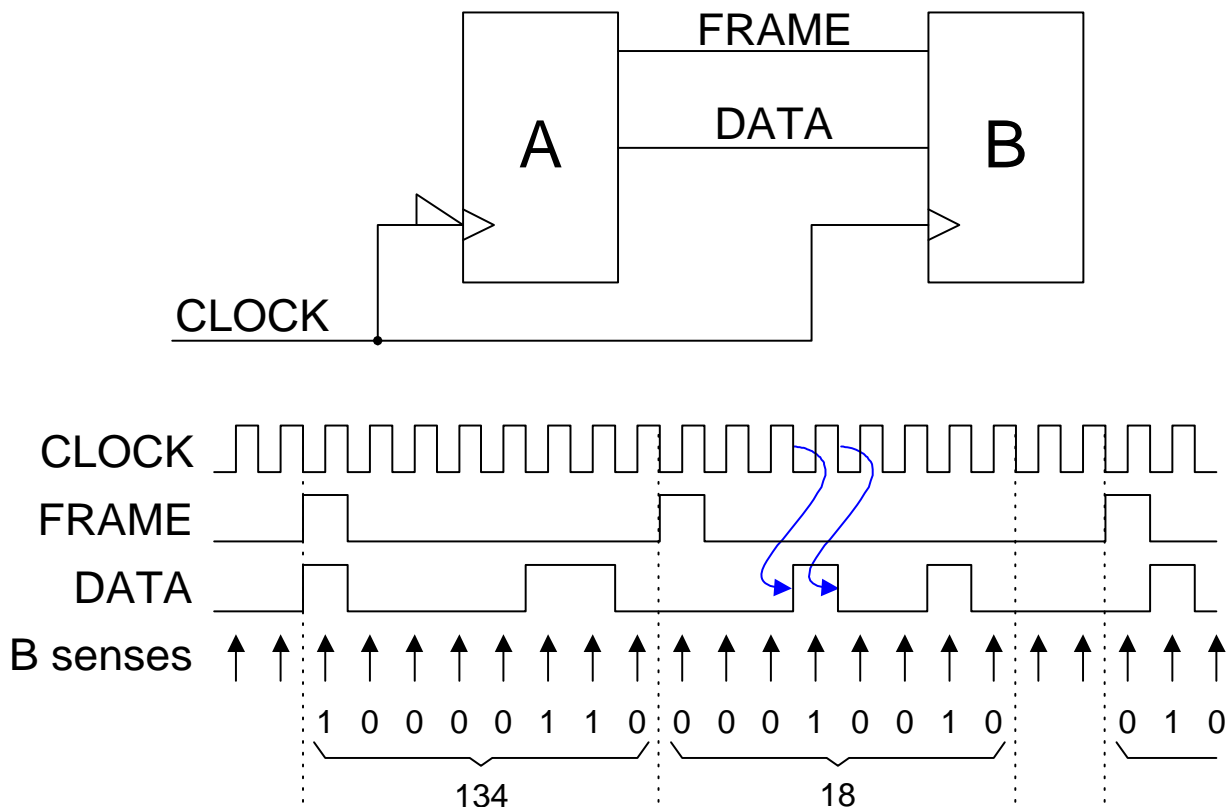
## Lecture 2

# Synchronous Bit-Serial Interfacing

### Objectives

- Explain how data is sent between two digital systems using a synchronous bit-serial protocol
  - **Synchronous**: same clock at transmitter & receiver
  - **Bit-serial**: Only one bit sent at a time
  - **Protocol**: The procedure for exchanging information
- Explain the meaning of setup and hold times
- Investigate the timing constraints in a transmission system

## Synchronous Bit-Serial Transmission



Transmitting 8 bit values from A to B:

- FRAME indicates the first bit of each value; the other 7 bits follow on consecutive clock cycles. The FRAME signal is often called a *frame sync* pulse.
- DATA changes on the *falling* CLOCK edge
- Propagation delays are often omitted from diagram.
- DATA is sensed by system B on the *rising* CLOCK edge to maximise tolerance to timing errors. We must always clock a flipflop at a time when its DATA input is not changing.

## Transmission Delays

Propagation speed =  $(L_0 C_0)^{-1/2}$  where  $L_0$  and  $C_0$  are inductance and capacitance per unit length.

For a uniform line this gives a total delay of  $(LC)^{1/2}$  where  $L$  and  $C$  are the total inductance and capacitance. Any additional load capacitance will increase delay.

Signal speed can be expressed in terms of:

- the speed of light ( $c = 30$  cm/ns)
- the geometry of the wiring
- the relative permittivity of the insulator:

### Examples:

- Coax cable:

$$c \times \epsilon_r^{-1/2} \Rightarrow 20 \text{ cm/ns for } \epsilon_r = 2.3 \text{ (teflon)}$$

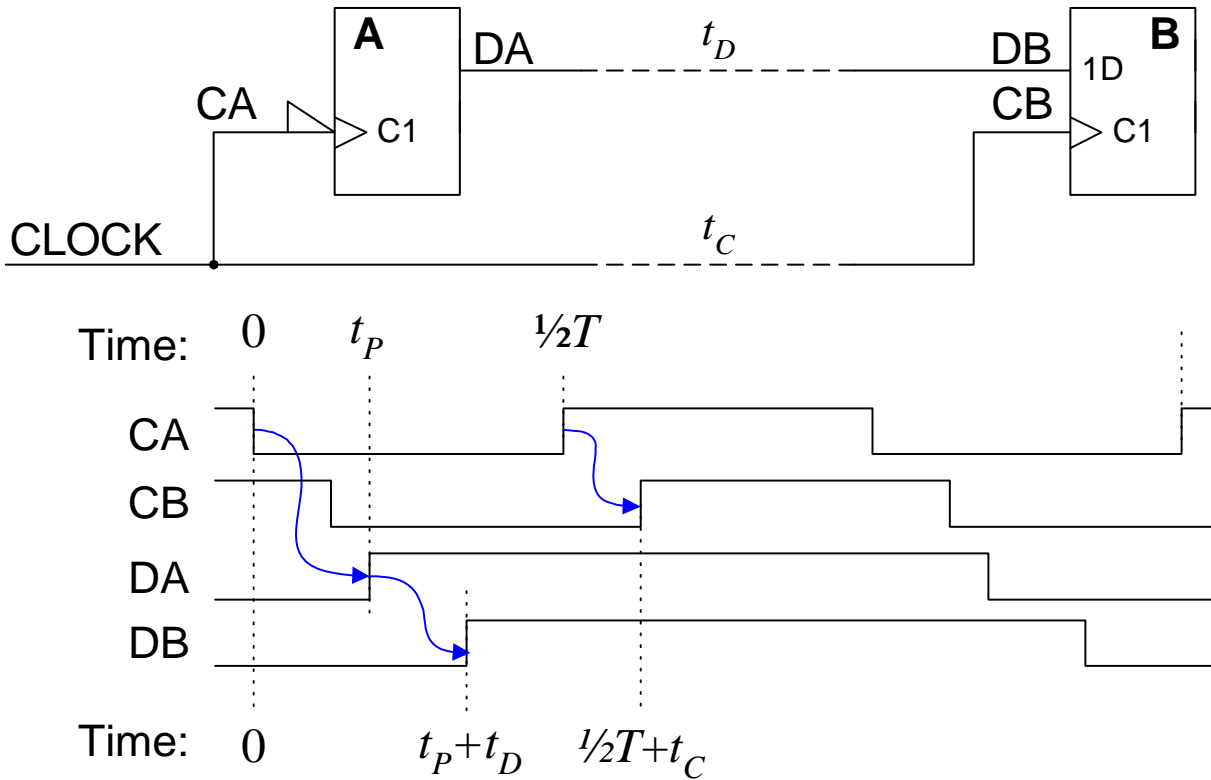
- PCB with ground plane:

$$1.4c \times (1.4 + \epsilon_r)^{-1/2} \text{ cm/ns} \Rightarrow 17 \text{ cm/ns for } \epsilon_r = 5 \text{ (fibreglass)}$$

### Rule-of-thumb:

Data travels along typical wires and circuit board tracks at about 15 cm/ns: half the speed of light.

## Timing Specifications



$t_p$       Propagation delay for device A.

$T$       Clock Period.

$t_C, t_D$       Transmission line delays for CLOCK and DATA

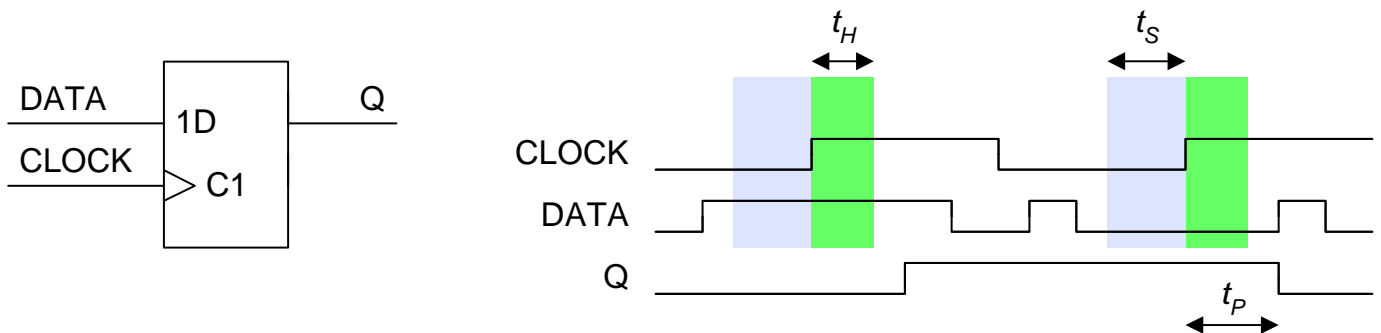
### For Device B:

- Data input changes at time  $t_p+t_D$
- Clock input changes  $\uparrow$  at time  $\frac{1}{2}T+t_C$



## Setup and Hold Times

The DATA input to a flipflop or register must not change at the same time as the CLOCK.



**Setup Time:** DATA must reach its new value at least  $t_S$  before the CLOCK $\uparrow$  edge.

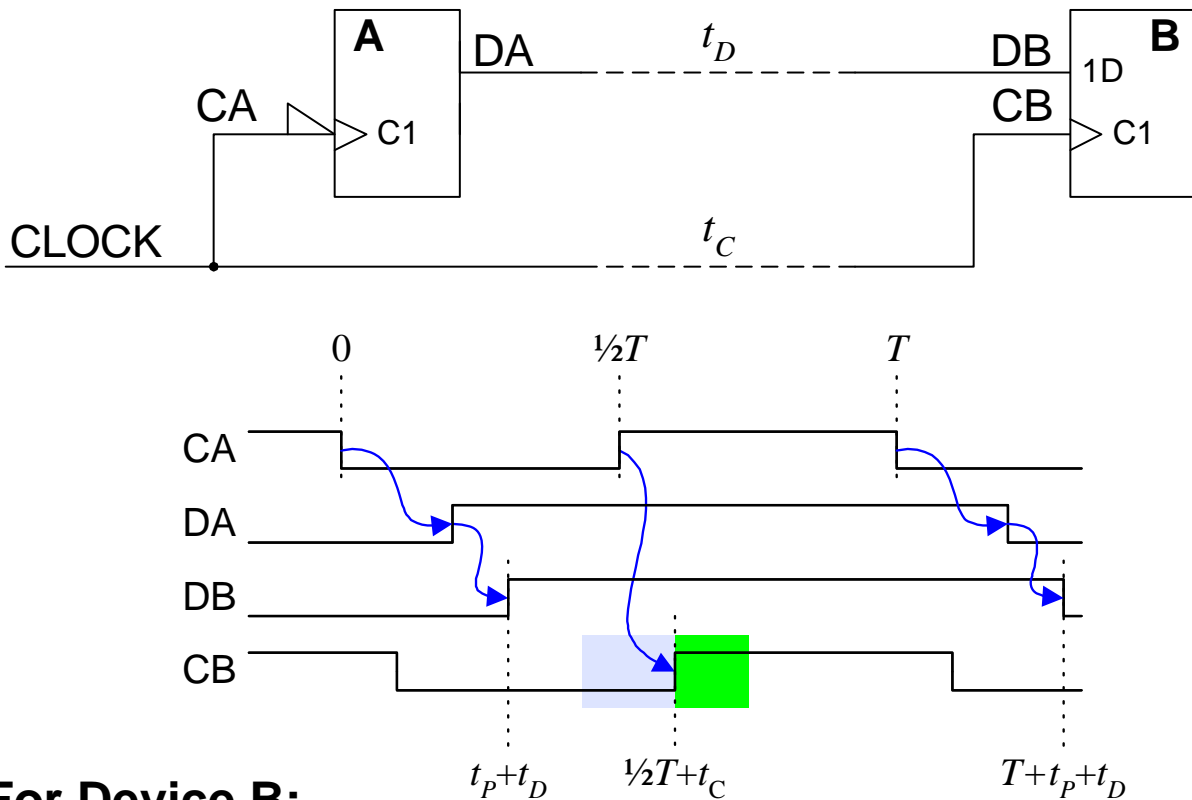
**Hold Time:** DATA must be held constant for at least  $t_H$  after the CLOCK $\uparrow$  edge.

Typical values for a register:  $t_S = 5 \text{ ns}$ ,  $t_H = 3 \text{ ns}$

The setup and hold time define a window around each CLOCK  $\uparrow$  edge within which the DATA **must not change**.

If these requirements are not met, the Q output may oscillate for many nanoseconds before settling to a stable value.

## Timing Constraints



### For Device B:

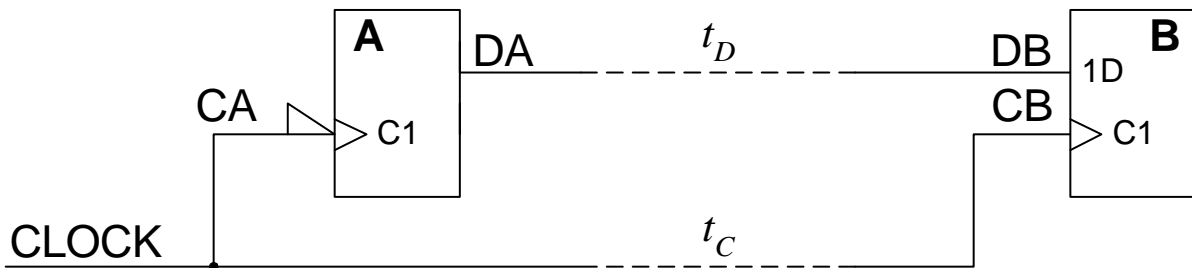
- Data input (DB) changes at  $t_p + t_D$  (and  $T + t_p + t_D$ )
- Clock $\uparrow$  (CB) at time  $\frac{1}{2}T + t_C$

### For reliable operation:

- Setup Requirement:  $t_p + t_D + t_S < \frac{1}{2}T + t_C$
- Hold Requirement:  $\frac{1}{2}T + t_C + t_H < T + t_p + t_D$

Get a pair of inequalities for each flipflop/register in a circuit.  
**You never get both  $t_S$  and  $t_H$  in the same inequality.**

## Example Values



For Motorola 56001 27MHz DSP processor:

$$0 < t_P < 50 \text{ ns}, t_S = 12 \text{ ns}, t_H = 27 \text{ ns}$$

Suppose differential delay:  $-10 < (t_D - t_C) < +10$

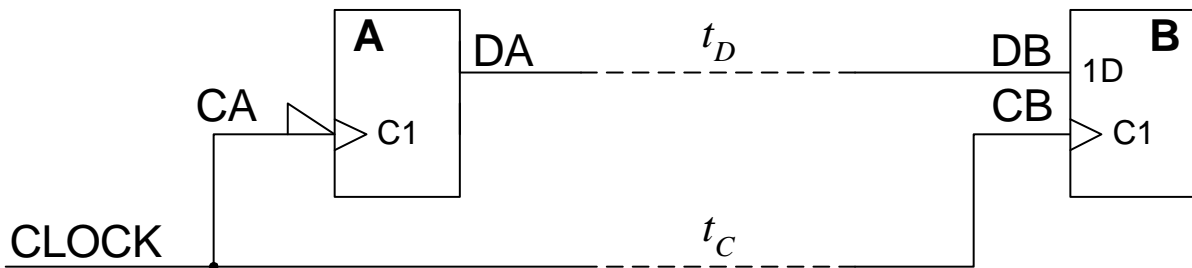
Find maximum CLOCK frequency (min CLOCK period):

- $\max(t_P + t_D) + t_S < \min(\frac{1}{2}T + t_C)$   
 $50 + 10 + 12 < \frac{1}{2}T + 0$  ( $t_D = 10, t_C = 0$ )  
 $\frac{1}{2}T > 12 + 50 + (+10) = 72$   $\Rightarrow T > 144 \text{ ns}$
- $\max(\frac{1}{2}T + t_C) + t_H < \min(T + t_P + t_D)$   
 $\frac{1}{2}T + 10 + 27 < T + 0 + 0$  ( $t_D = 0, t_C = 10$ )  
 $\frac{1}{2}T > 27 + 10 = 37$   $\Rightarrow T > 74 \text{ ns}$

Hence  $f_{\text{CLOCK}} < 1/144 = 7 \text{ MHz}$

To test for worst case: make the left side of the inequality as big as possible and the right side as small as possible.

## Propagation Delay Constraint Inequalities



### When do they arise

- Whenever a flipflop's clock and data input signals originate from the same ultimate source. Here CB and DB both originate from CLOCK. You normally get two inequalities for each flipflop in a circuit.

### Relationship between setup and hold inequalities:

- Setup Requirement:  $t_P + t_D + t_S < \frac{1}{2}T + t_C$
- Hold Requirement:  $\frac{1}{2}T + t_C + t_H < t_P + t_D + T$
- To get the Hold inequality you change  $t_S$  to  $t_H$ , swap the sides of the other terms and add  $T$  onto the right side.

### Are both $t_S$ and $t_H$ ever in the same inequality?

- No.

### How do you decide to take the max or the min?

- For a  $<$ , take max of everything on the left and min of everything on the right.
- max = most positive: for example,  $\max(-7, -2) = -2$

## Quiz Questions

1. What is a *bit-serial* transmission system?
2. What is a *synchronous* transmission system?
3. In a synchronous transmission system in which the transmitted data changes on the rising edge of the CLOCK, why is it normal for the receiver to sense the data on the falling edge of the CLOCK ?
4. What is the purpose of the *frame sync* signal In a synchronous bit-serial transmission system?
5. How far does a signal travel along a typical wire in one nanosecond?
6. What do the terms *setup time* and *hold time* mean?
7. Why do you get a pair of timing inequalities for each flipflop or register in a circuit?
8. In formulating the timing inequalities, how do you choose what to use for a quantity whose value may lie anywhere within a particular range?

Answers are all in the notes.

## Lecture 3

# Asynchronous Bit-Serial Interfacing

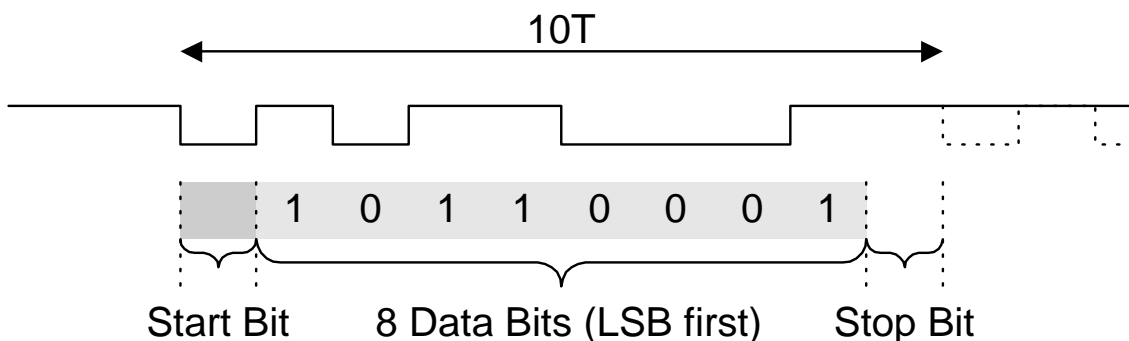
### Objectives

- Explain how data is sent between two digital systems using an asynchronous bit-serial protocol
- Explain why it is necessary to include START and STOP bits in an asynchronous protocol.
- Explain the circuitry needed for an asynchronous bit-serial receiver
- Derive the tolerances for the transmitter and receiver clocks in an asynchronous bit-serial system

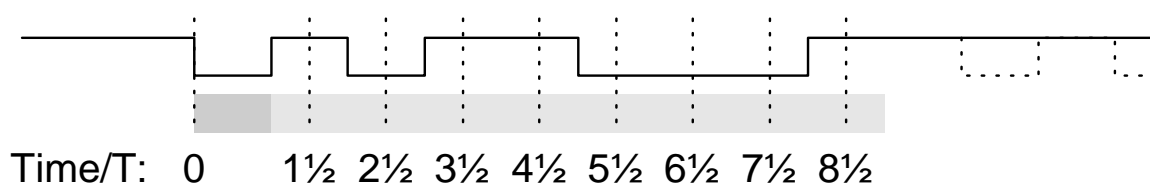
## Asynchronous Bit-Serial Transmission

Combine timing and data into a single signal to circumvent differential delays over long distances (saves wires too).

### RS232: Serial Port on a PC

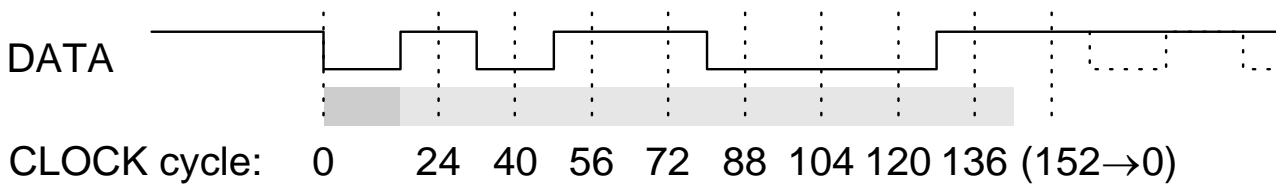


- Idle state has signal = 1. START bit indicates new byte.
- Data transmitted LSB first (above example equals  $141_{10}$ )
- Bit cell duration is  $T$ : e.g.  $T=52 \mu\text{s}$ ,  $1/T = 19200$  baud
- STOP bit needed to ensure signal goes to 1 before the next START bit which might follow immediately.
- Signal is decoded by sampling each bit in the centre of its cell an appropriate time after the start bit:

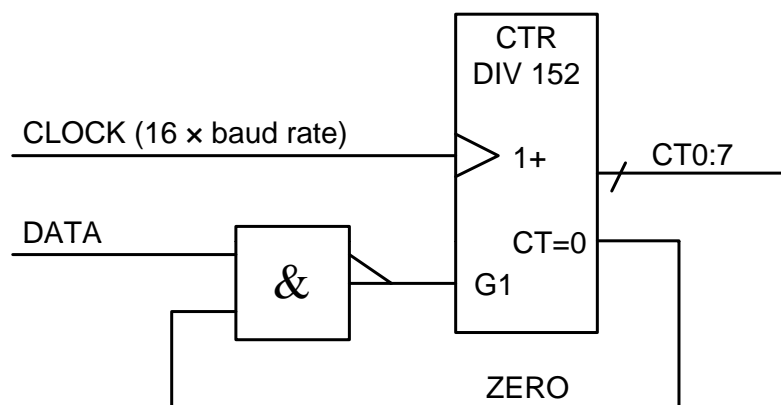


## RS232 Receiver Timing

Good time resolution  $\Rightarrow$  use a master clock period of  $T/16$ :



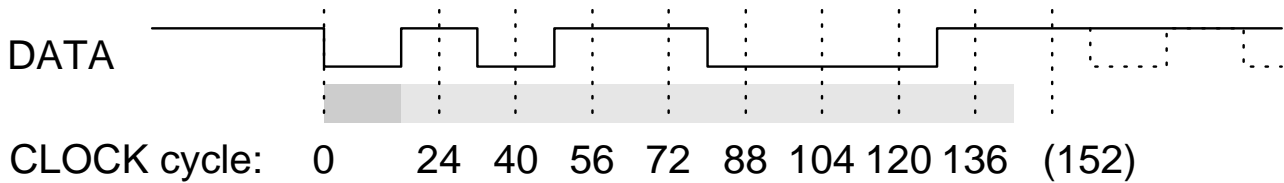
- Middle of STOP bit is after  $9\frac{1}{2}$  bitcells  $\Rightarrow 9\frac{1}{2} \times 16 = 152$  master clock cycles.
- Use  $\div 152$  counter but hold it at 0 until the START bit arrives  $\Rightarrow$  inhibit counting whenever  $CT=0$  and  $DATA=1$ .
- Counter will increment either if  $DATA=0$  or if  $CT$  is already non-zero.
- We use a *clock enable* input,  $G1$ , to control whether or not the counter increments; much better design than using gates to modify the clock signal.



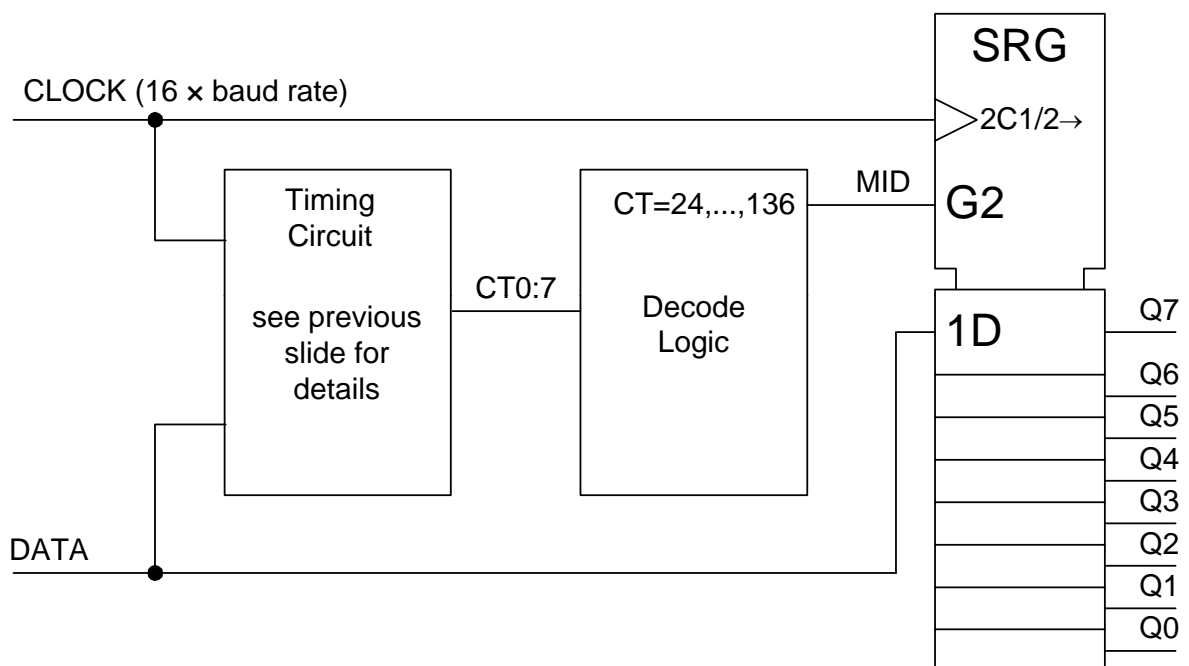
The  $CT=0$  output from counter goes high when the contents of the counter,  $CT$ , are zero. Generate this signal using a NOR gate connected to all 8 counter outputs.



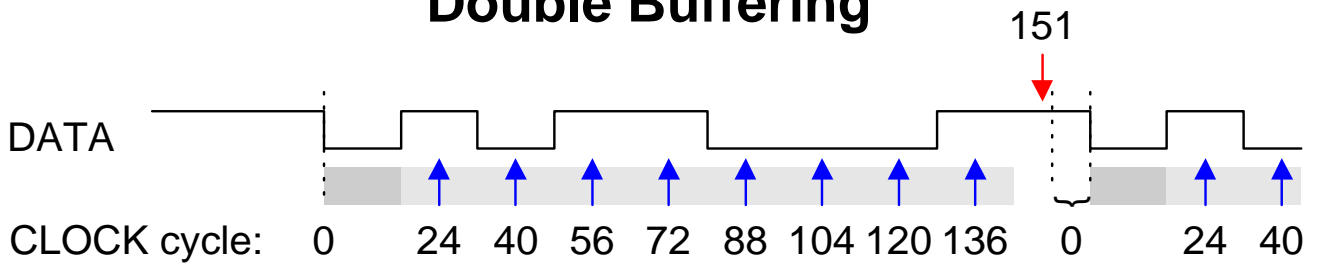
## RS232 Receiver



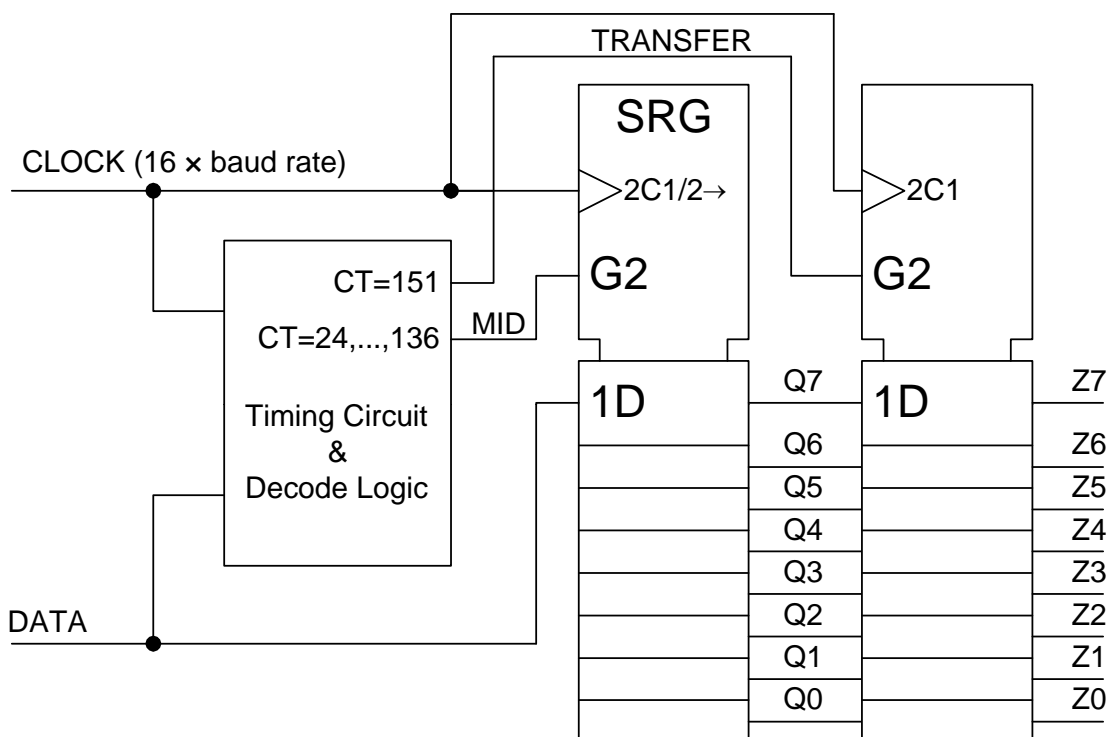
- Use an 8-bit shift register to store the data value. Only allow it to shift when  $CT = 24, 40, \dots, 136$ .
- The decode logic output, MID, goes high when the counter has one of these values (all odd multiples of 8  $\Rightarrow$  four LSBs =  $1000_2$ ).
- Notation:
  - The shift register clock has two functions separated by a /:  $2C1$  clocks first bit,  $2 \rightarrow$  shifts the rest.
  - Both these functions are controlled by the *clock enable* input, G2.



## Double Buffering



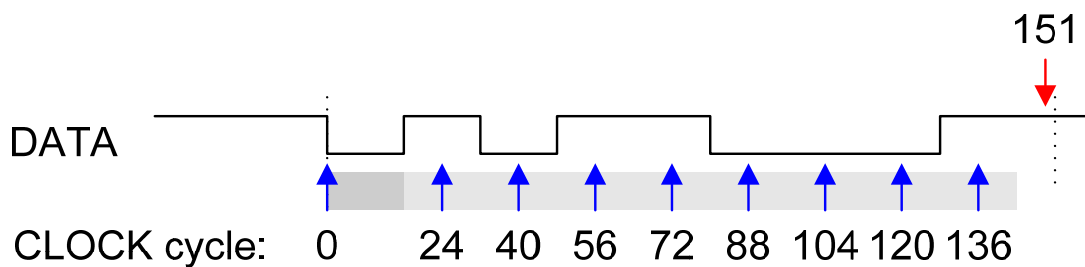
- The 8 data bits only stay in the shift register for  $3T$  before they get shifted out again by the next data byte.
- Host microprocessor must respond to an interrupt within this time and retrieve the data.
- Use a second register to grab the data at  $T=151$  and keep it for a whole  $10T$ . This gives the  $\mu P$  more time.



# Timing Errors

- **Ideal situation:**

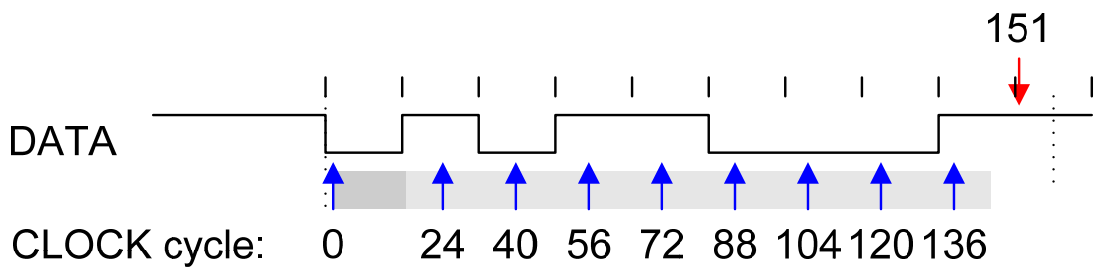
- Receiver clock period  $P = T/16$
- Counter starts counting exactly on DATA falling edge



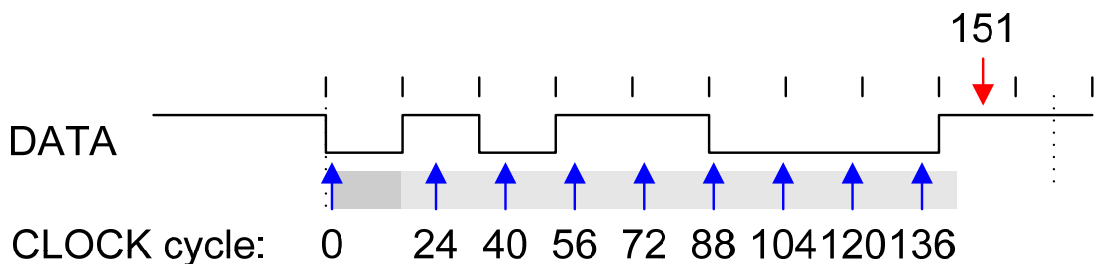
- **Real situation:**

- Receiver clock period not exactly  $T/16$
- Counter starts with some delay
  - On first rising edge of  $P$  after DATA goes low

P slightly too small

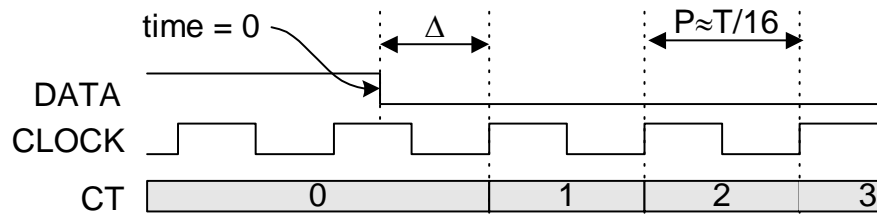


P much too small



## RS232 Receiver Timing

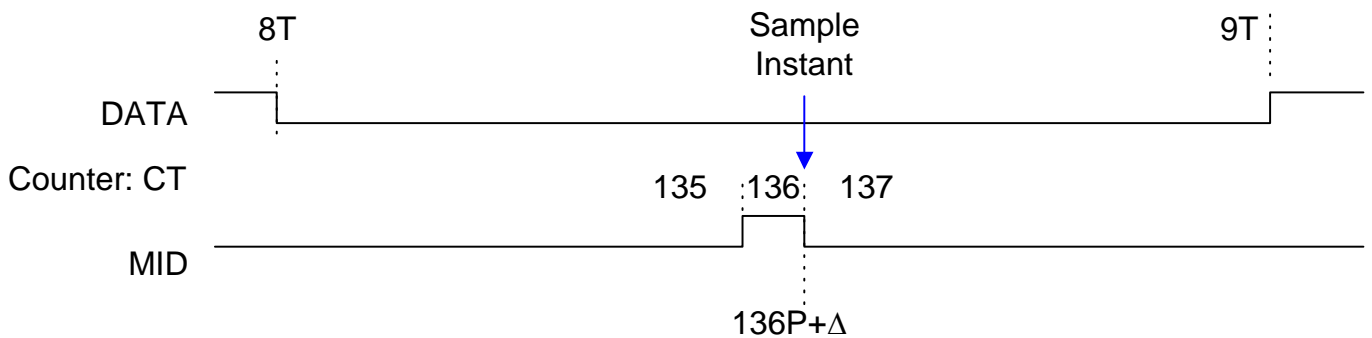
CT will change to 1 on the first CLOCK  $\uparrow$  edge after DATA goes to 0:



Neglecting logic propagation delays,  $0 < \Delta < P$  where  $P$  is receiver clock period.

- Count  $0 \rightarrow 1$  a time  $\Delta$  after the START bit
- Count  $n \rightarrow n+1$  a time  $nP + \Delta$  after the START bit

### Timing in the last (MSB) bit cell:



We will sample the correct bit cell if:  $8T < 136P + \Delta < 9T$

$$8T < 136P + 0 \quad \Rightarrow \quad T/P < 17$$

$$136P + P < 9T \quad \Rightarrow \quad T/P > 15.2$$

Hence  $T/P = 16 + 6.3\% - 5.0\%$  which implies a clock accuracy of around  $\pm 2.5\%$  at transmitter and receiver.

## Quiz Questions

1. How can you be sure that in the RS232 protocol there will always be a high-to-low transition at the beginning of each transmitted byte ?
2. What is the function of the *clock enable* input on a counter or register?
3. What logic gate is needed to detect when the contents of a counter is equal to zero?
4. If an asynchronous protocol has one START bit, eight data bits and one STOP bit, how many bitcell periods is it from the beginning of the START bit until the centre of the STOP bit?
5. What is the function of an input pin that is labelled  $2C1/2 \rightarrow$ ?
6. If the CLOCK input of a counter has period  $P$ , what is the range of possible delays between the counter's enable pin going high and the counter incrementing?
7. What is the purpose of double buffering the data in an asynchronous bit-serial receiver?
8. How can you tell if a binary number is an odd multiple of 16?

Answers are all in the notes.

## Lecture 4

# Microprocessor to Memory Interface

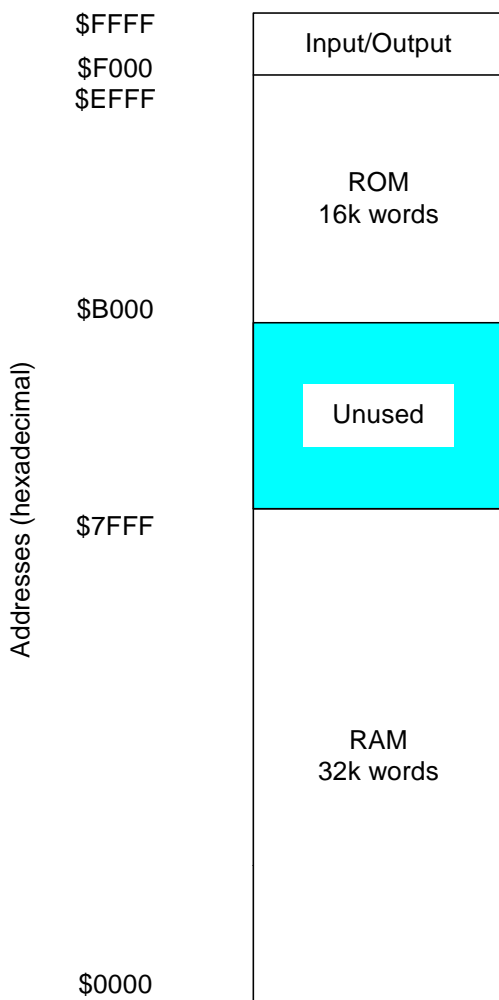
### Objectives

- Explain how memory is connected to a microprocessor
- Describe the sequence of events in reading from and writing to a static RAM
- Describe the structure and input/output signals of a static RAM

# Microprocessor Memory Map

A typical 8-bit microprocessor has

- A 16-bit address bus, A15:0
  - Can have up to  $2^{16}=65536$  memory locations
  - Value is usually written in hexadecimal often with \$ prefix:
    - e.g. \$1000 =  $2^{12} = 4k = 4096$
- An 8-bit data bus, D7:0
  - Each data word in memory has  $2^8 = 256$  possible values

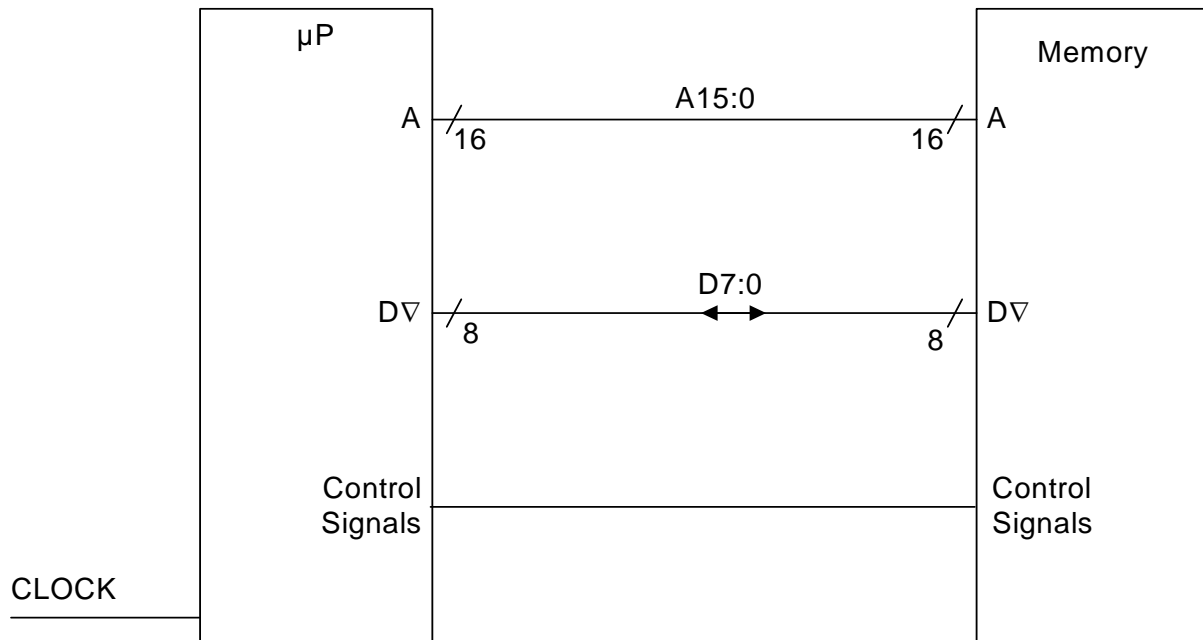


We can tell which region of memory an address is in by inspecting the top few bits:

A15:12		
F:	1111	Input/Output
E:	1110	ROM
D:	1101	ROM
C:	1100	ROM
B:	1011	ROM
A:	1010	
9:	1001	
8:	1000	
7:	0111	RAM
6:	0110	RAM
5:	0101	RAM
4:	0100	RAM
3:	0011	RAM
2:	0010	RAM
1:	0001	RAM
0:	0000	RAM

INOUT =  $A_{15} \cdot A_{14} \cdot A_{13} \cdot A_{12}$   
 ROM =  $A_{15} \cdot A_{14} \cdot \neg(A_{13} \cdot A_{12}) + A_{15} \cdot \neg A_{14} \cdot A_{13} \cdot A_{12}$   
 RAM =  $\neg A_{15}$

## Microprocessor ↔ Memory Interface

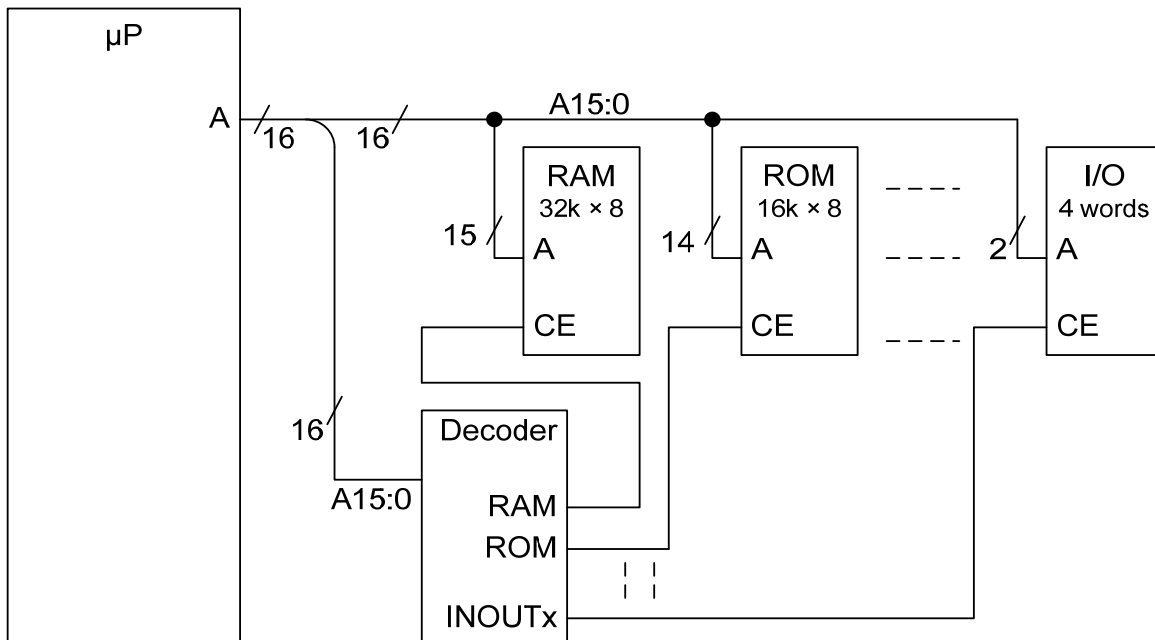


During each memory cycle:

- A15:0 selects one of  $2^{16}$  possible memory locations
- D7:0 transfer one word (8 bits) of information either to the memory (write) or to the microprocessor (read).
- D7:0 connections to the microprocessor are tri-state ( $\nabla$ ): they can be:
  - “logic 0”, “logic 1” or “high impedance” (inputs)
- The control signals tell the memory what to do and when to do it.



## Memory Chip Selection

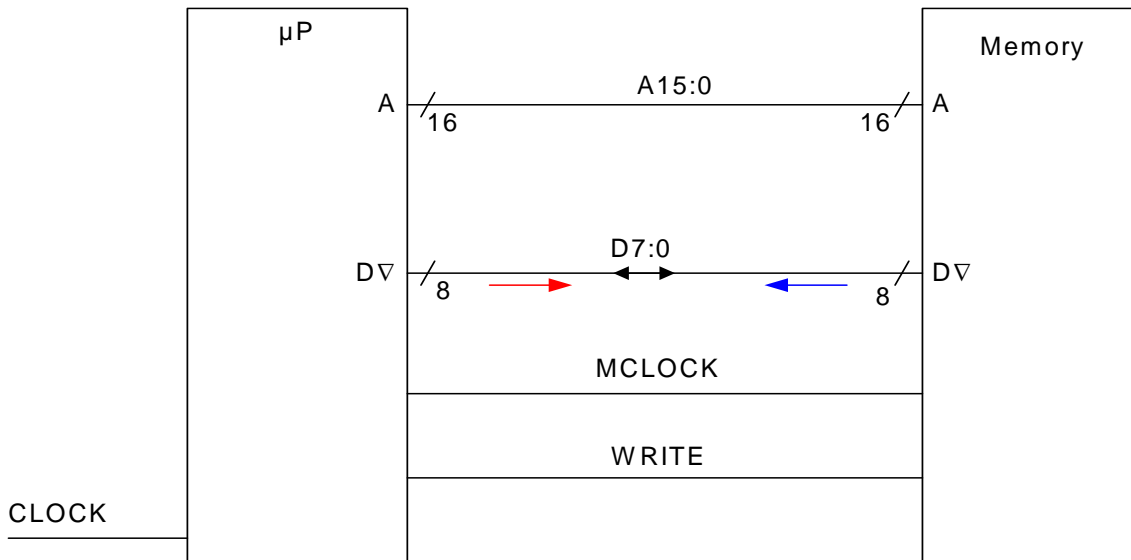


- Each memory circuit has a “chip enable” input (CE)
- The “Decoder” uses the top few address bits to decide which memory circuit should be enabled. Each one is enabled only for the correct address range:
 
$$\text{RAM} = \neg A_{15}$$

$$\text{ROM} = A_{15} \cdot (A_{14} \cdot \neg(A_{13} \cdot A_{12}) + \neg A_{14} \cdot A_{13} \cdot A_{12})$$

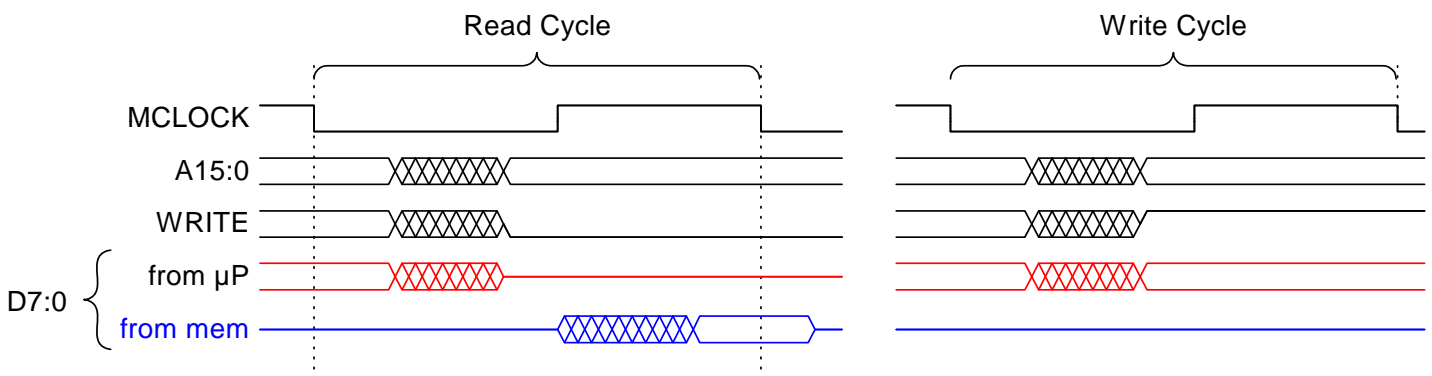
$$\text{INOUT}_x = A_{15} \cdot A_{14} \cdot A_{13} \cdot A_{12} \cdot \neg A_{11} \cdot A_{10} \cdot \neg A_9 \cdot A_8 \cdot \neg A_7 \cdot A_6 \cdot A_5 \cdot A_4 \cdot \neg A_3 \cdot A_2$$
- INOUT<sub>x</sub> responds to addresses: \$F574 to \$F577  
other I/O circuits will have different addresses
- Low  $n$  address bits select one of  $2^n$  locations within each memory circuit (value of  $n$  depends on memory size)

# Memory Interface Control Signals



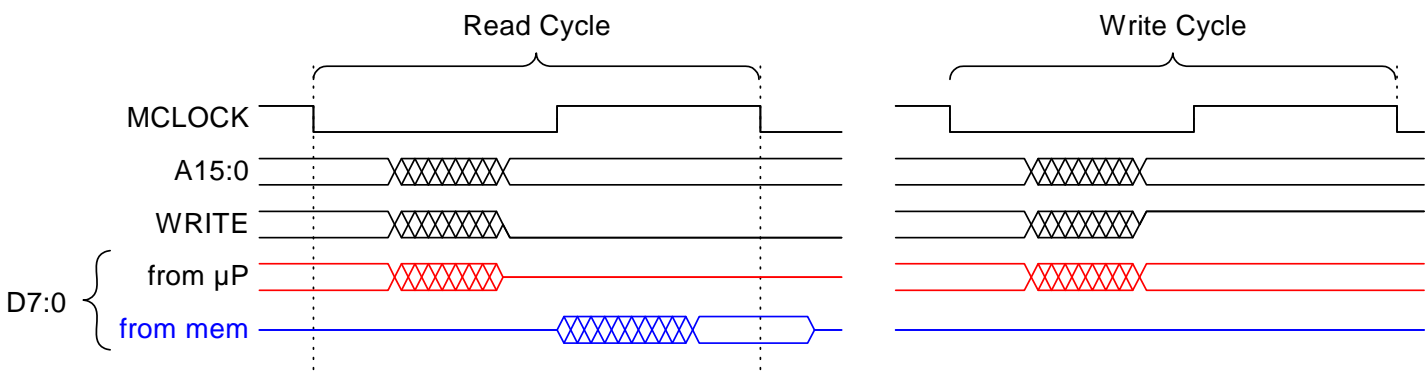
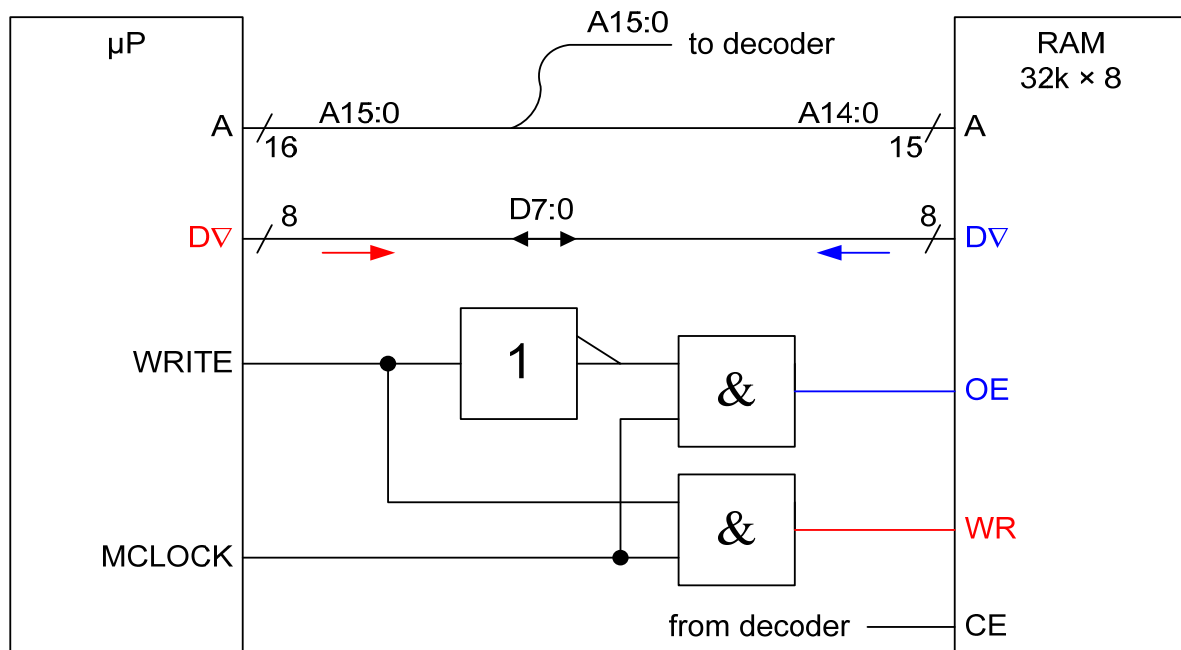
Control signals vary between microprocessors but all have:

- A clock signal to control the timing (can be the same as the system CLOCK)
- A signal to say whether the microprocessor wants to read from memory or to write to memory
  - Must make sure that D7:0 is only driven at one end



**D7:0 from memory** only allowed when MCLOCK·!WRITE true

# Memory Circuit Control Signals



- **Output enable:**  $OE = MCLOCK \cdot \overline{WRITE}$  turns on the D7:0 output from the memory
- **Write enable:**  $WE = MCLOCK \cdot WRITE$  writes new information into the selected memory location with data coming from microprocessor
- **Chip enable:** comes from the decoder and makes sure the memory only responds to the correct addresses

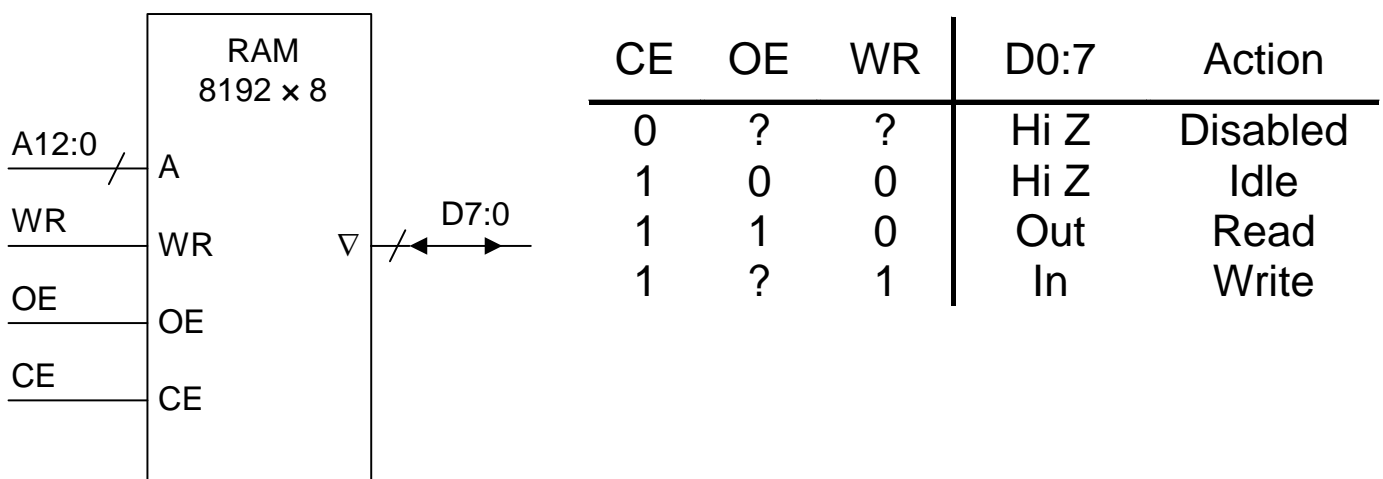
## RAM: Read/Write Memory

Static RAM: Data stored in bistable latches

Dynamic RAM: Data stored in charged capacitors:  
retained for only 2ms.

Less circuitry  $\Rightarrow$  denser  $\Rightarrow$  cheaper.

### 8k x 8 Static RAM



▽ Tri-state output: Low, High or Off (High Impedance).  
Allows outputs from several chips to be connected;  
Designer must ensure only one is enabled at a time.

CE Chip Enable: disabling chip cuts power by 80%.

OE Output Enable: Turns the tri-state outputs on/off.

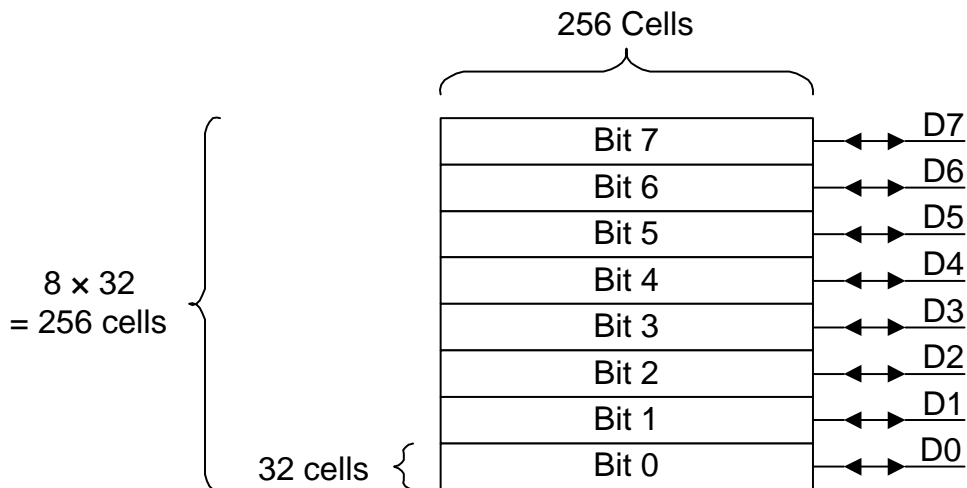
A12:0 Address: selects one of the  $2^{13}$  8-bit locations.

WR Write: stores new data in selected location

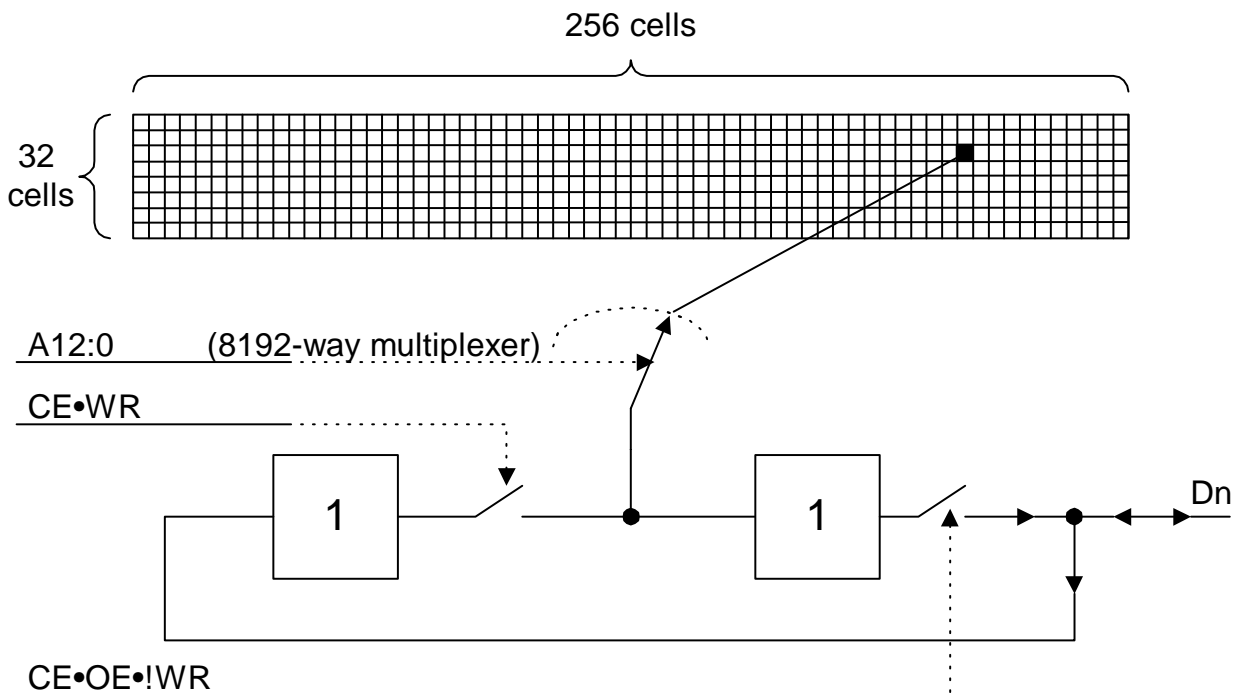
D7:0 Data in for write cycles or out for read cycles.

## 8k x 8 Static RAM

The 64k memory cells are arranged in a square array:



For each output bit, an 8192-way multiplexer selects one of the cells. The control signals, **OE**, **CE** and **WR** determine how it connects to the output pin via buffers:



Occasionally DIN and DOUT are separate but  $\Rightarrow$  more pins

## Quiz Questions

1. What is the *memory map* of a microprocessor system
2. Why do all microprocessor systems include some read-only memory (ROM)
3. What does it mean if a digital device has a *tri-state* output? When are such outputs necessary ?
4. What is the difference between the *chip enable* and the *output enable* inputs of a static RAM?
5. If a static RAM has  $n$  address inputs and  $m$  data outputs, how many bits of information does it store?
6. What is the binary value of the three most significant address bits for the hexadecimal address \$BC37 ?

Answers are all in the notes.

## Lecture 5

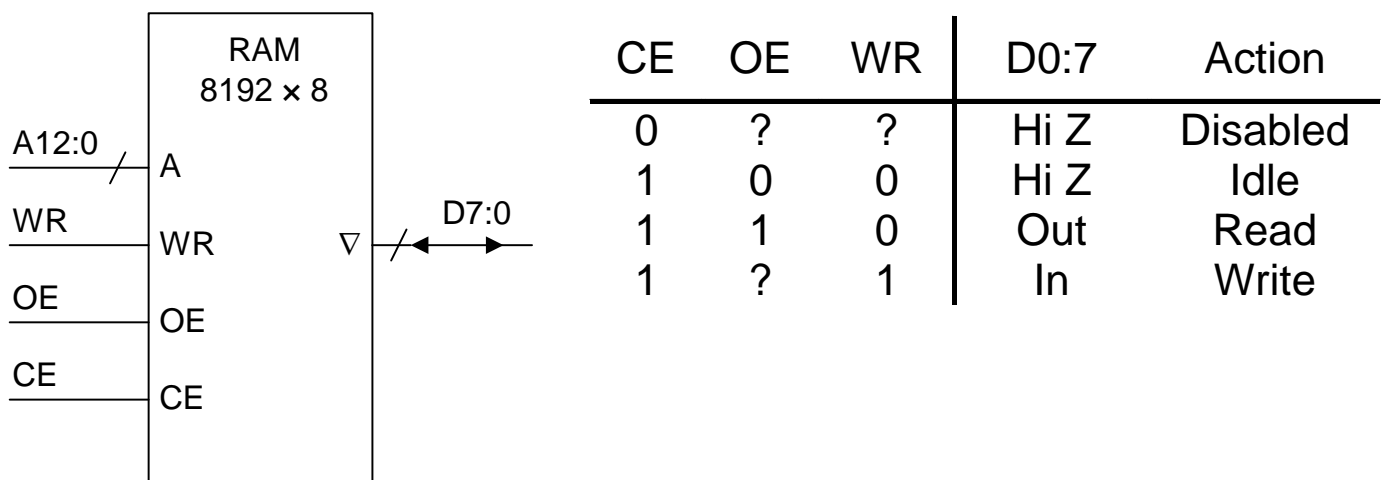
# Microprocessor to Memory Interface

### Objectives

- Investigate the timing constraints for a microprocessor when reading from or writing to memory.

## RAM: Read/Write Memory

### 8k x 8 Static RAM



▽ Tri-state output: Low, High or Off (High Impedance). Allows outputs from several chips to be connected; Designer must ensure only one is enabled at a time.

A12:0 Address: selects one of the  $2^{13}$  8-bit locations.

D7:0 Data in for write cycles or out for read cycles.

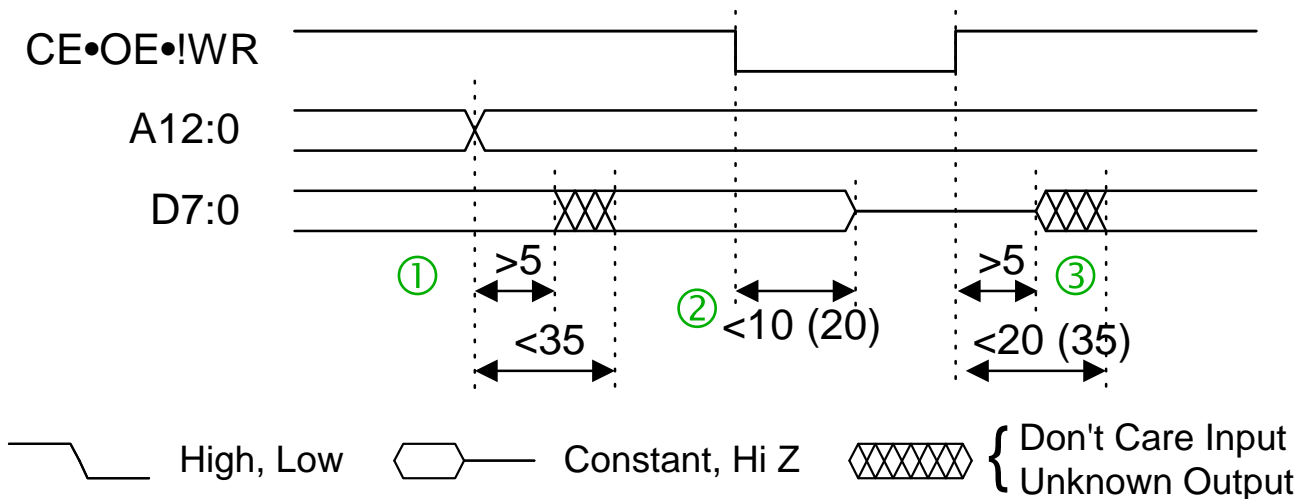
CE **Chip Enable**: disabling chip cuts power by 80%.

OE **Output Enable**: Turns the tri-state outputs on/off.

WR **Write**: stores new data in selected location



# Memory Read Cycle



Note: Time axis not to scale

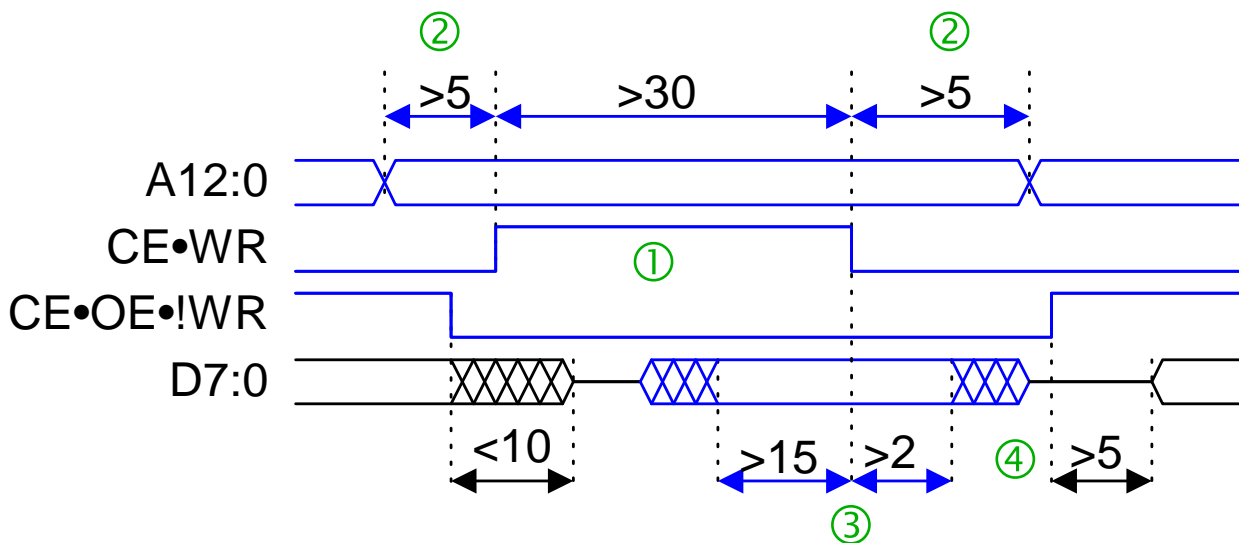
A *read cycle* happens when  $CE \cdot OE \cdot \overline{!WR}$  is true.

- ① If  $A_{12:0}$  changes,  $D_{7:0}$  remains for at least 5 ns and goes to new value within 35 ns. Rubbish in between even if new and old locations contain the same value.
- ② If a read cycle ends due to  $OE$  going low, the outputs go  $Hi-Z$  within 10 ns
- ③ If a read cycle starts due to  $OE$  going high,  $D_{7:0}$  stays  $Hi-Z$  for at least another 5 ns and the selected word appears within 20ns

You can use  $CE$  instead of  $OE$  but it is slower: 20 ns to turn off and 35 ns to turn on (in parentheses on timing diagram).

When reading data, the propagation delay to the  $D_{7:0}$  outputs is called the RAM's *access time*: 35 ns from  $A_{12:0}$  and 20 ns from  $OE$ .

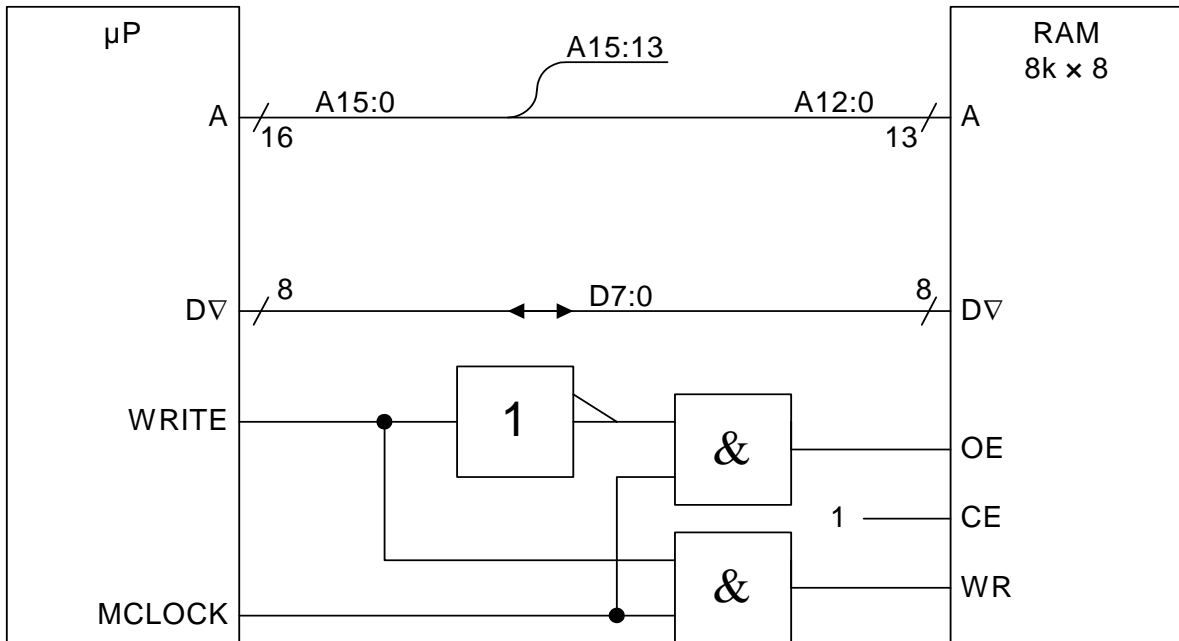
## Memory Write Cycle



A *write cycle* happens whenever CE•WR is true.

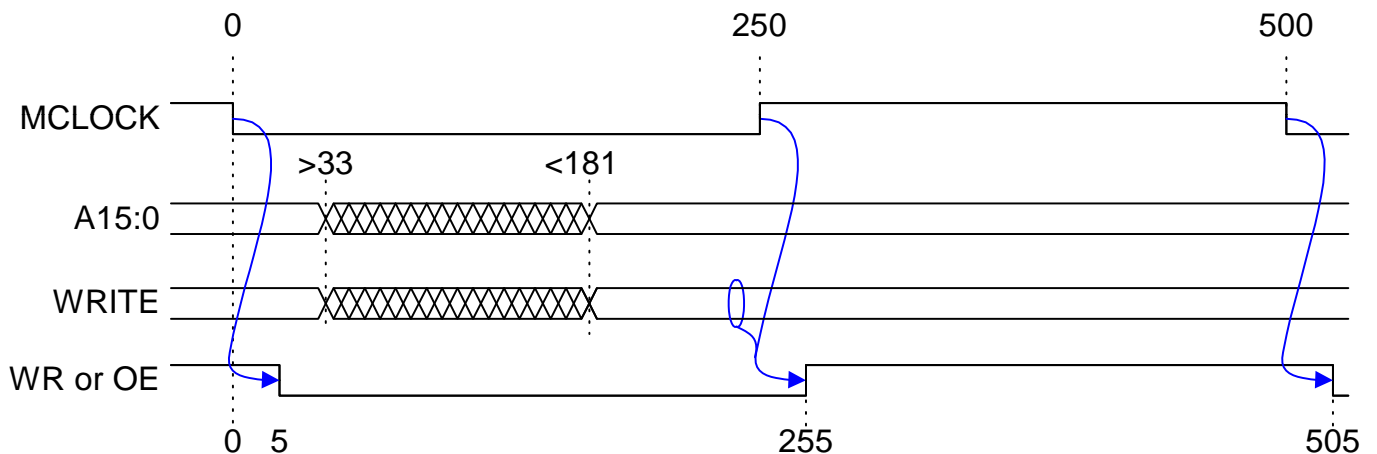
- ① CE•WR must go high for at least 30 ns.
  - ② To avoid writing to unwanted locations, the address, A12:0 must remain constant for at least 5 ns at both ends of the write pulse.
  - ③ Input data D0:7 only matters at the *end* of the write pulse. Setup & hold times of 15 ns & 2 ns define a window within which it must not change.
  - ④ Input When CE•OE•!WR goes high, the memory reverts to read mode. The input data must be removed from D7:0 before this happens.
- Timing specifications that end on an output are guarantees from the chip manufacturer (shown in black).
  - Timing specifications that end on an input are requirements that the designer must meet (shown blue).

## Microprocessor ↔ Memory Interface

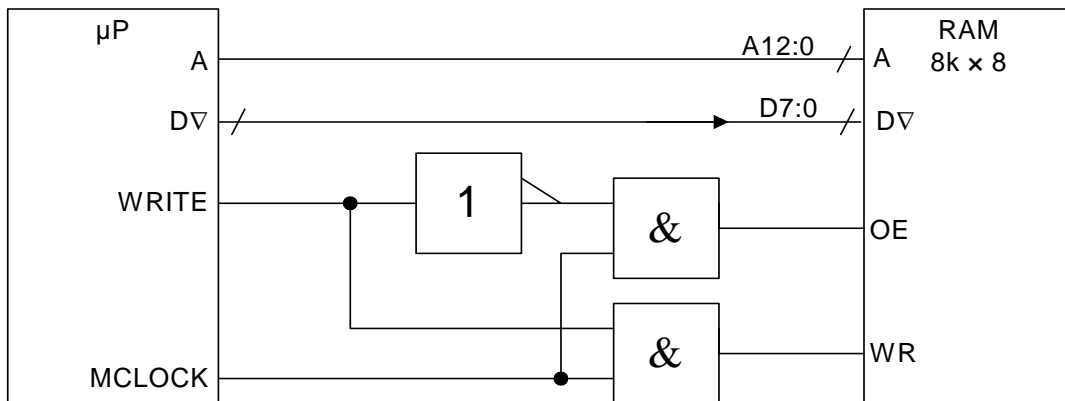
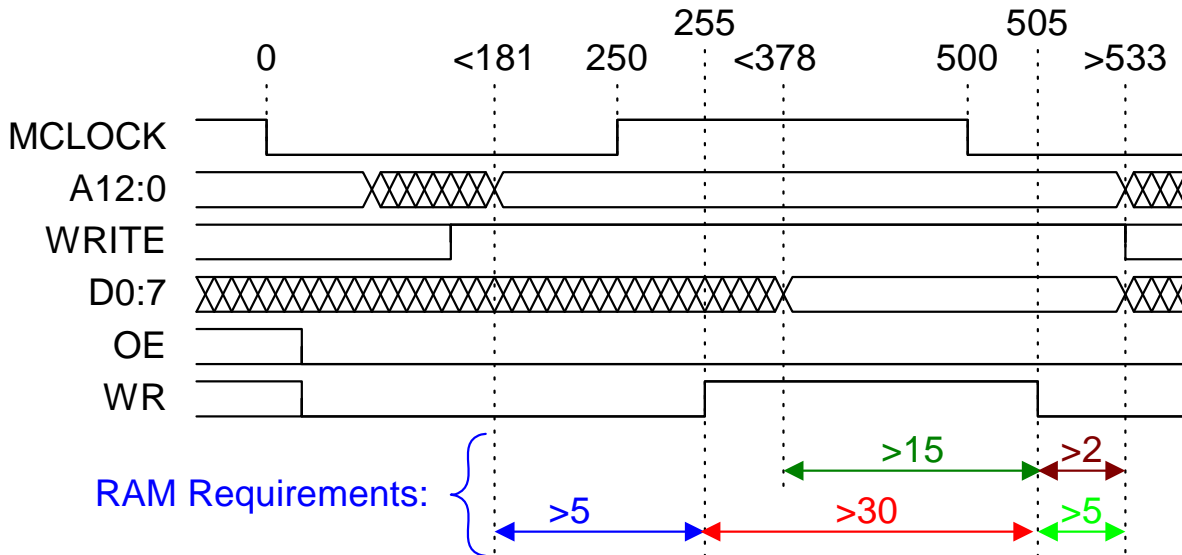


$$OE = MCLOCK \cdot !WRITE \quad WR = MCLOCK \cdot WRITE$$

- Reading or writing takes place during the **second half of the clock cycle** when MCLOCK is high.
- WRITE output from  $\mu P$  determines whether WR or OE goes high. Assume NAND gate delay = 5 ns.

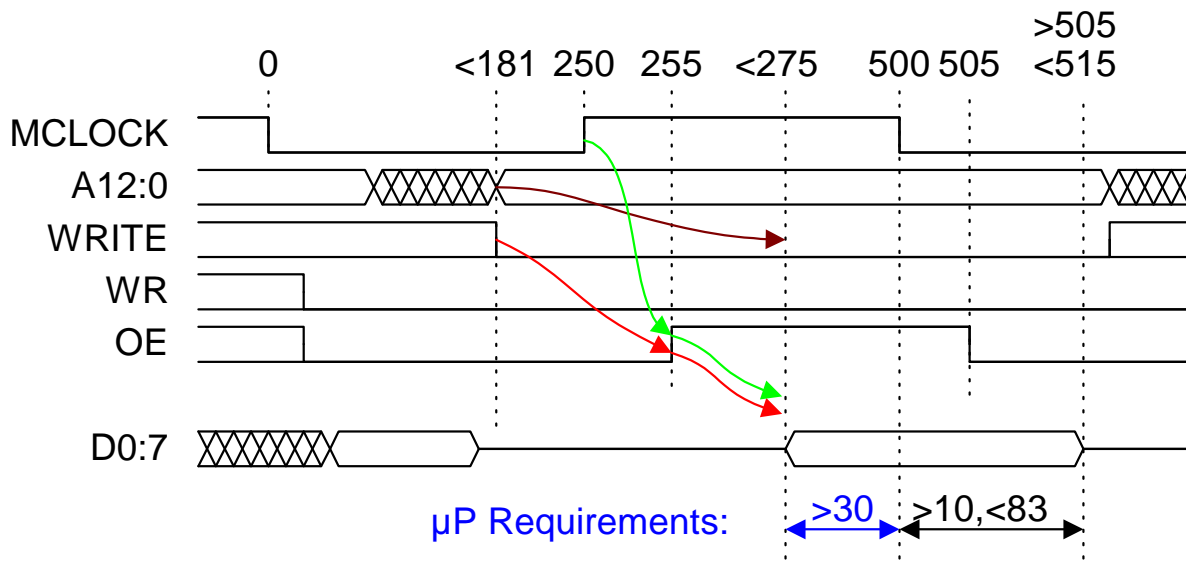


## Microprocessor Write to Memory

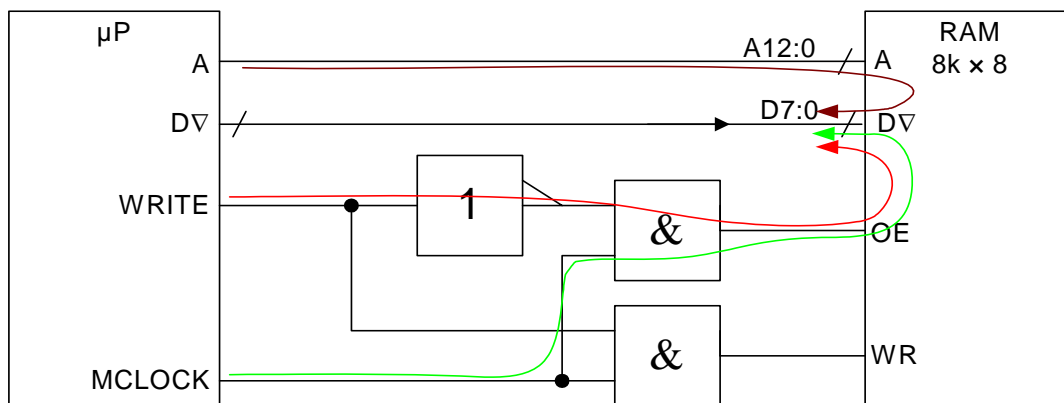


- μP emits data within 128ns of MCLOCK↑
- Requirements:
  - Addr→WR setup:  $181+5 < 255$  ✓
  - Data →!WR setup:  $378+15 < 505$  ✓
  - WR pulse:  $255+30 < 505$  ✓
  - Addr hold:  $505+5 < 533$  ✓
  - Data hold:  $505+2 < 533$  ✓

# Microprocessor Read Setup Time

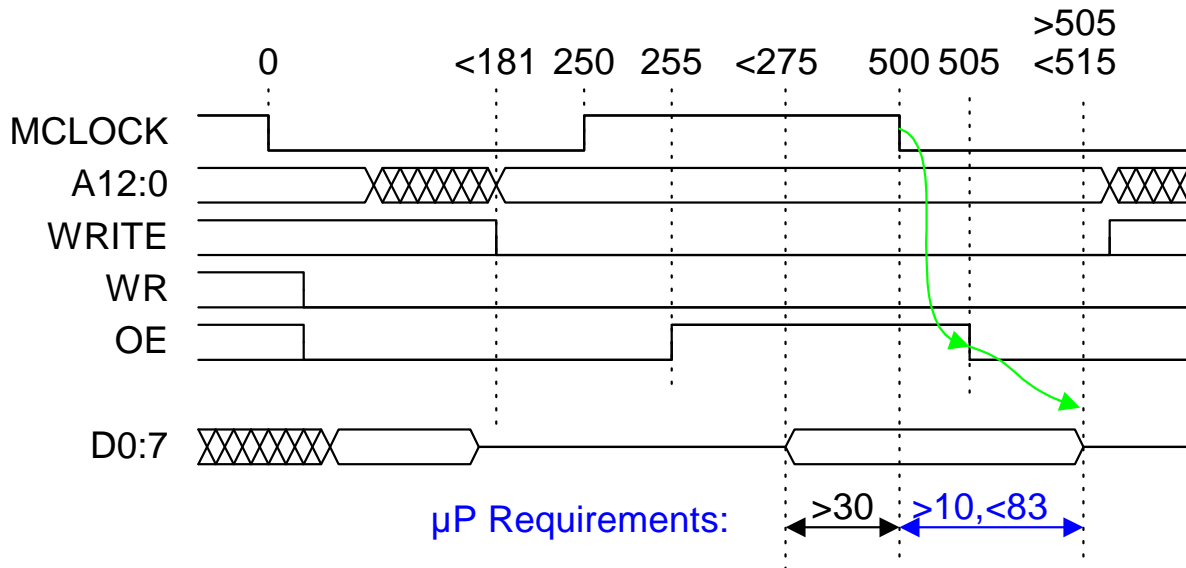


- Data Setup time: 30 ns before MCLOCK↓.
  - Three paths must be satisfied
  - Check each one individually

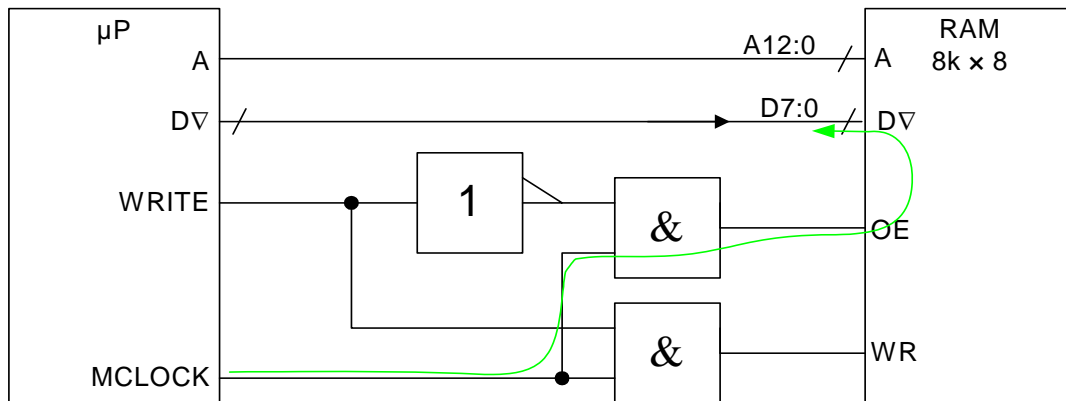


- Requirements:
  - Addr to Data setup:  $181+35+30 < 500$  ✓
  - WRITE to Data setup:  $181+5+5+20+30 < 500$  ✓
  - MCLOCK to Data setup:  $250+5+20+30 < 500$  ✓

# Microprocessor Read Hold Time



- D7:0 must go tristate 10 to 83 ns after MCLOCK↓.
  - MCLOCK path is the only relevant one
  - OE↓ to tristate delay varies between 0 and 10 ns



- Requirements:

- Min hold:  $505 > 500 + 10$



- Max hold:  $515 < 500 + 83$



May need to add some delay to !OE signal to meet min hold

## Quiz Questions

1. What is the *access time* of a static RAM?
2. When writing to a static RAM, why is does the state of the data inputs matter only at the end of the write pulse?
3. How do you check timing constraints if the manufacturer specifies a maximum propagation delay but no minimum ?
4. How do you check timing constraints if the validity of an output depends on several of the input signals ?

Answers are all in the notes.

## Lecture 6

# Control Logic

### Objectives

- Understand how digital systems may be divided into a data path and control logic
- Appreciate the different ways of implementing control logic
- Understand how shift registers and counters can be used to generate arbitrary pulse sequences
- Understand the circumstances that give rise to output glitches



## Control Logic

Most digital systems can be divided into

- **Data Path**: adders, registers etc
- **Control Logic**: generates timing signals to ensure things happen at the right time and in the right order

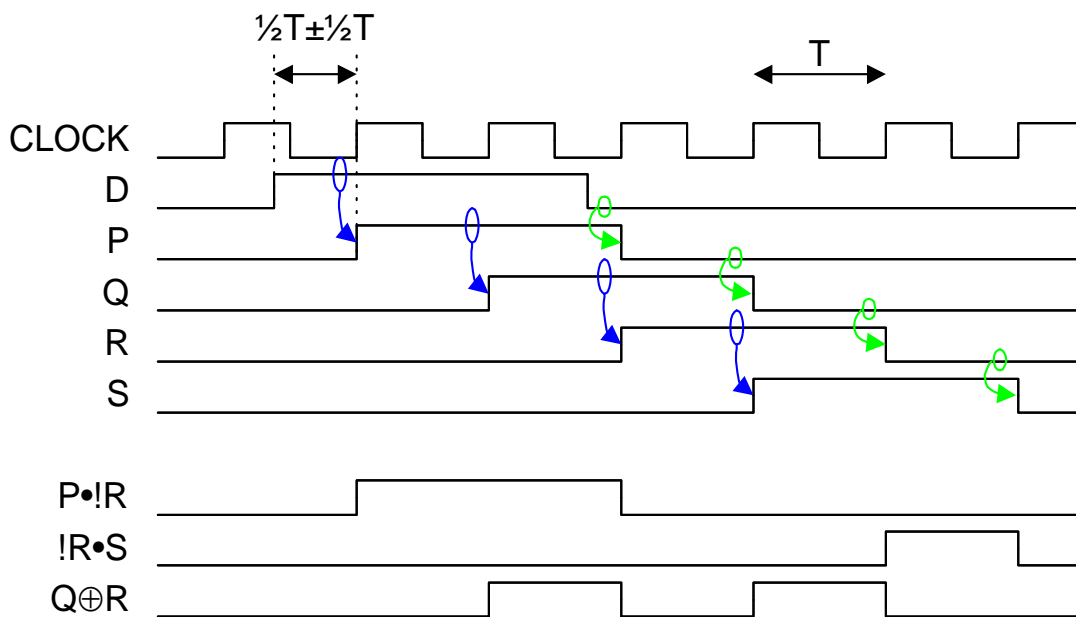
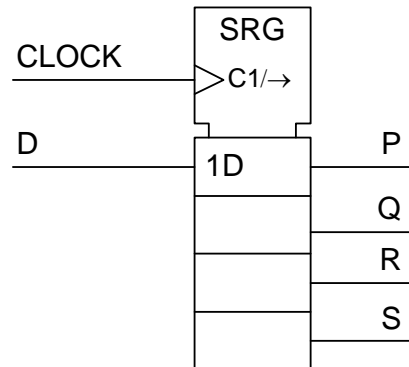
Control logic can be implemented with:

- **Microprocessor/Microcontroller**
  - + Cheap, very flexible, design easy (software)
  - Slow: most actions require >20 instructions = 2  $\mu$ s @ clock speed of 10 MHz.  
Use for slow applications.
- **Synchronous State Machine**
  - + Fast (20 ns/action), Cheap using programmable logic.
  - Hard to design complex systems. Limited data storage.  
Use for fast, moderately complex systems.
- **Counters/Shift Registers**
  - + Fast, Cheap, Very easy design.
  - Simple systems only.  
A special case of synchronous state machines.  
Use for very simple systems (fast or slow).

## Shift Registers

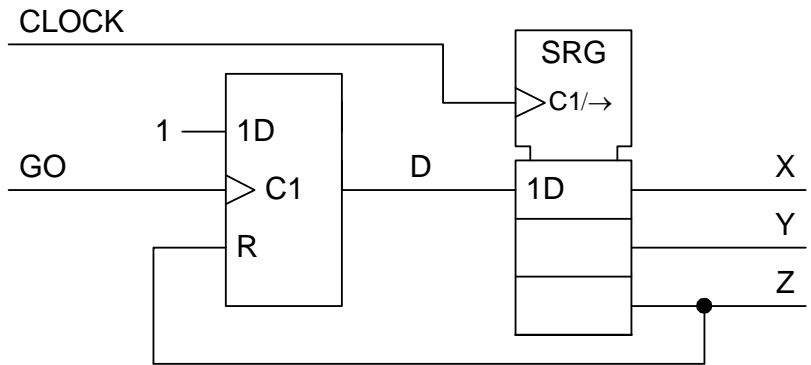
Easy way to make a sequence of events happen in response to a trigger:

- P, Q, R and S are delayed versions of D but with all transitions on the CLOCK  $\uparrow$
- Delay from D to P is between 0 and 1 clock cycle.

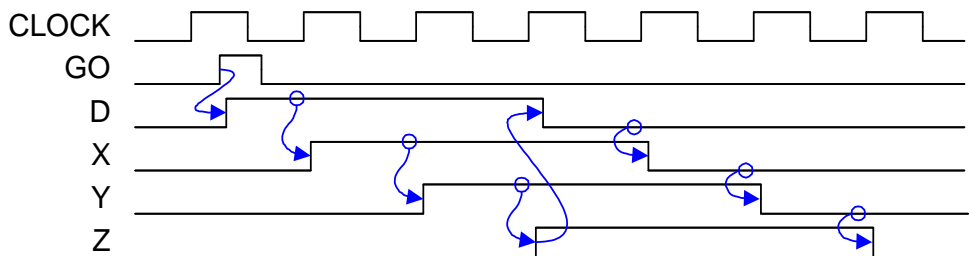


- $P \cdot \neg R$  gives pulse of length  $2T$  approx  $\frac{1}{2}T$  after  $D \uparrow$ .
- $\neg R \cdot S$  gives pulse of length  $T$  approx  $2\frac{1}{2}T$  after  $D \downarrow$ .
- $Q \oplus R$  gives pulses of length  $T$  approx  $1\frac{1}{2}T$  after  $D \uparrow$  &  $\downarrow$

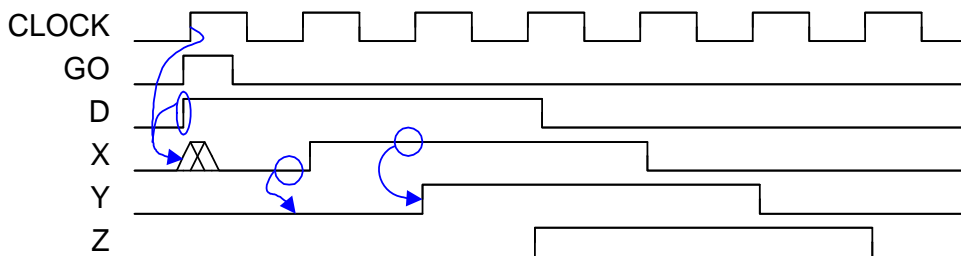
## Shift Registers with Short Input Pulses



- D might be ignored if it lasts < 1 CLOCK period
- GO input is sent to a edge-triggered input
- Works like a toaster: Z causes D to turn off halfway through the whole cycle.



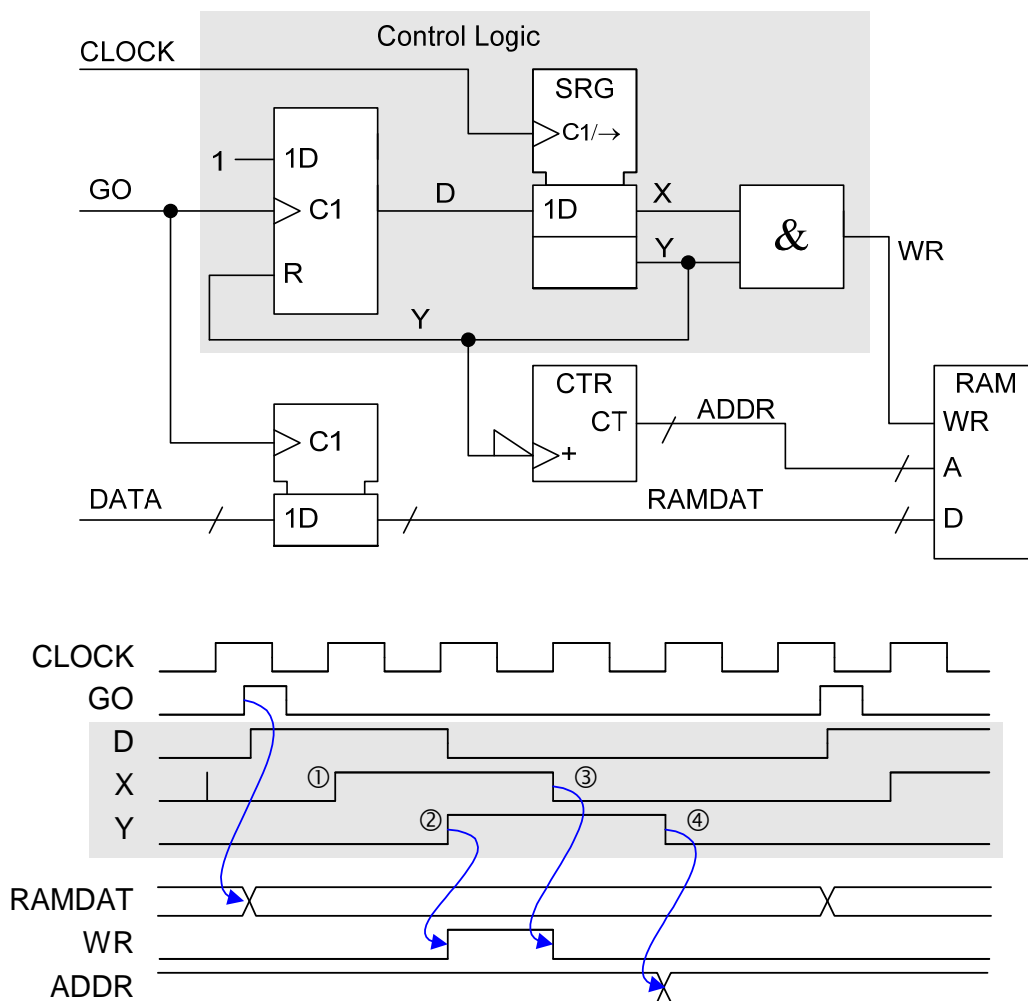
- Use the X output with care: it may oscillate for tens of ns if D changes within setup/hold window:



- X is OK by next clock ↑ so Y and Z are safe to use.

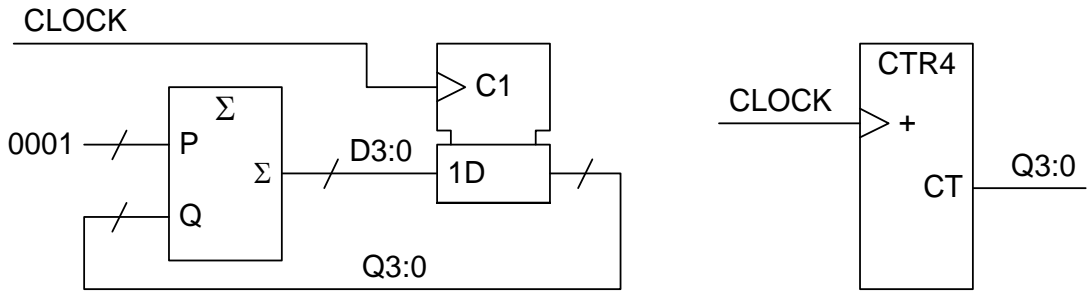
## Shift Register Example: Logic Analyser

On every GO rising edge we must sample DATA and store it in the RAM.

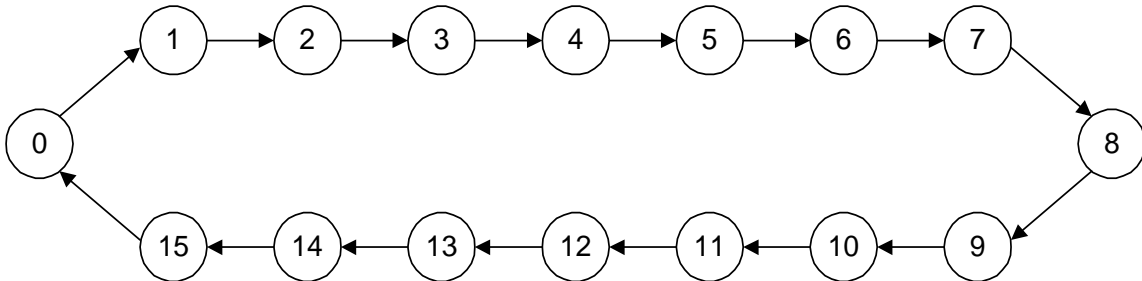


- RAM control signals are easily generated from the shift register. Four time instants available: ① to ④.
- We don't use ① so it doesn't matter if X has a glitch on the previous cycle since it is ANDed with Y (which is low at the time).

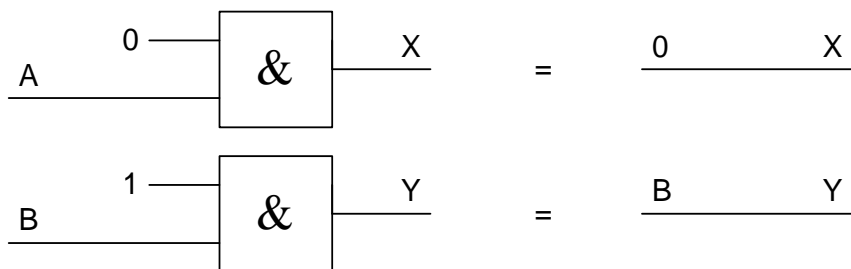
## Synchronous Counters



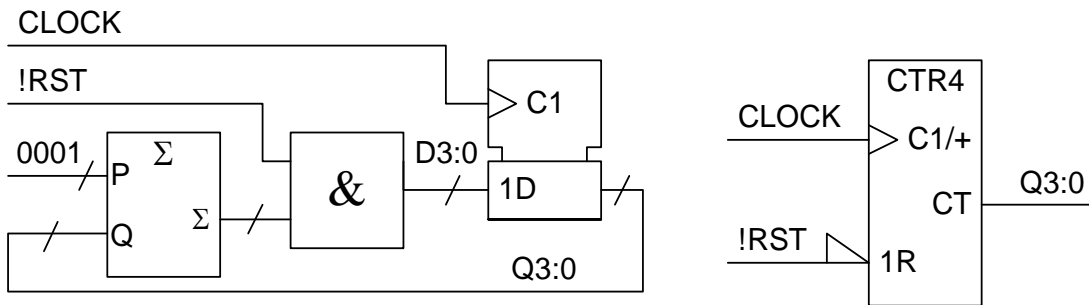
- An  $N$  bit binary counter has a cycle length of  $2^N$  states. We can draw a state diagram in which one transition is made for each clock  $\uparrow$  :



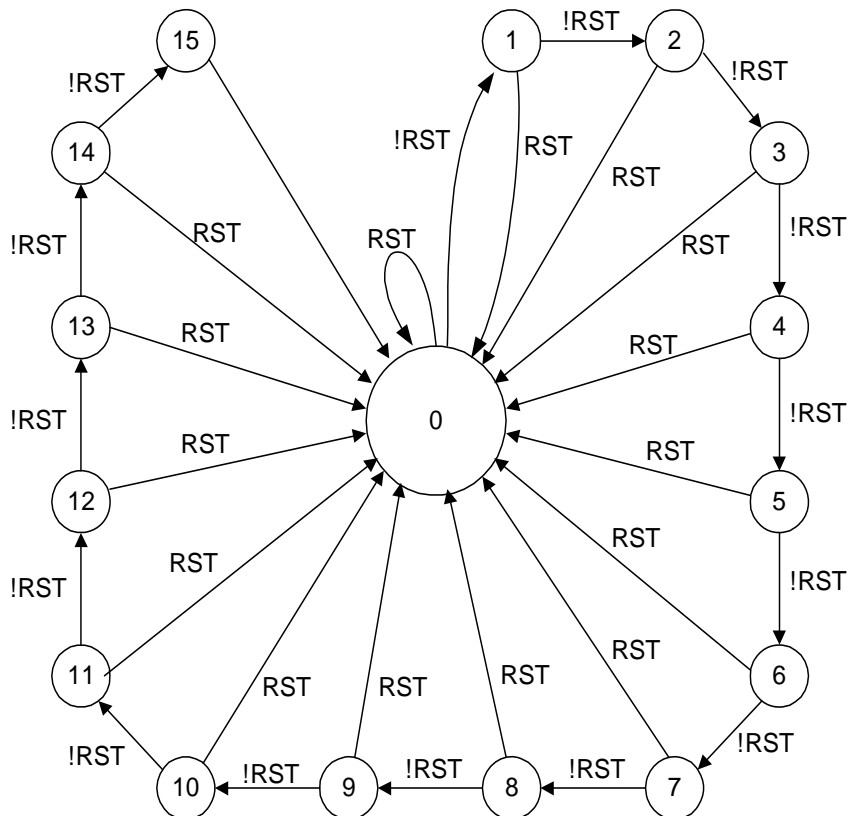
- Adder can be simplified: one set of inputs is fixed so many gates can be eliminated:



## Synchronous RESET



- This is a *synchronous* reset input: taking  $\overline{!RST}$  low has no effect until the next clock  $\uparrow$
- In a synchronous counter everything is done by manipulating the D inputs of the register.



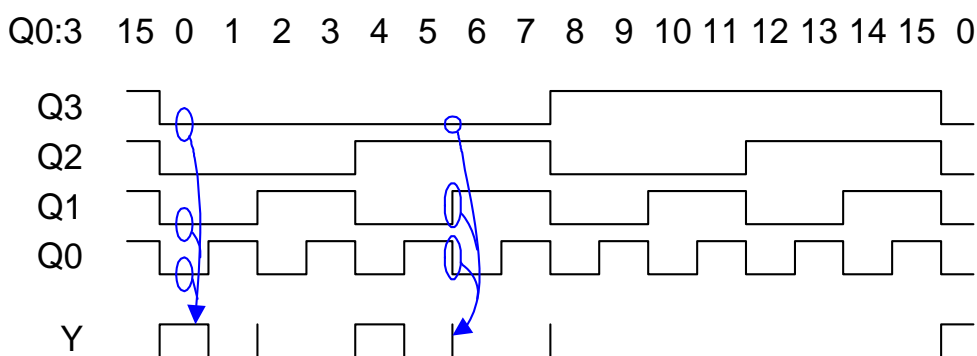
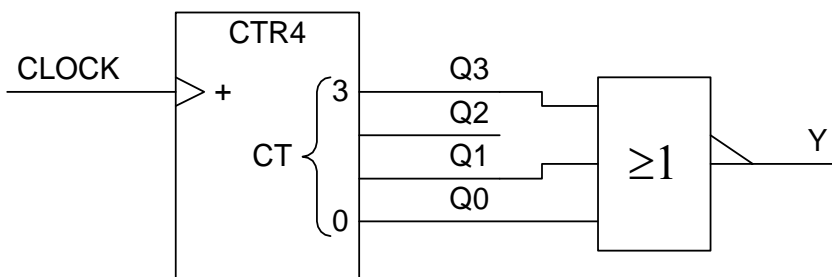


## Output Glitches

If  $k$  counter bits change “simultaneously”, other logic circuits using them may briefly see any of  $2^k$  possible values.

Glitches are possible at the logic circuit output if both:

1. These  $2^k$  values include any that would cause the logic circuit output to change.
- and 2. The logic circuit output is meant to remain at a constant value.



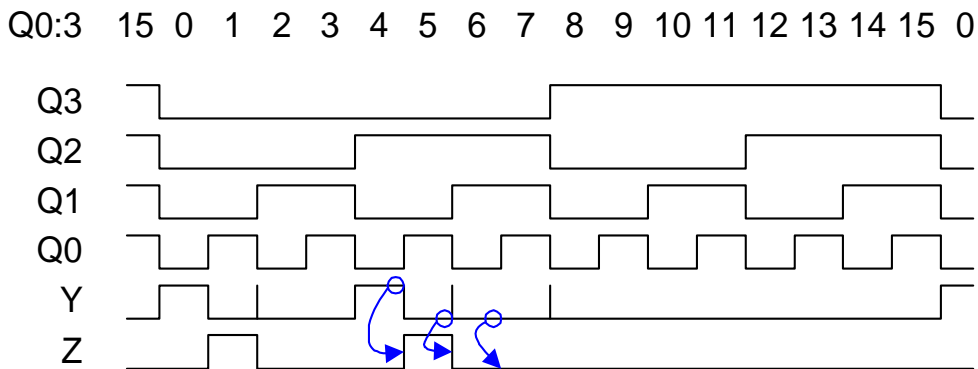
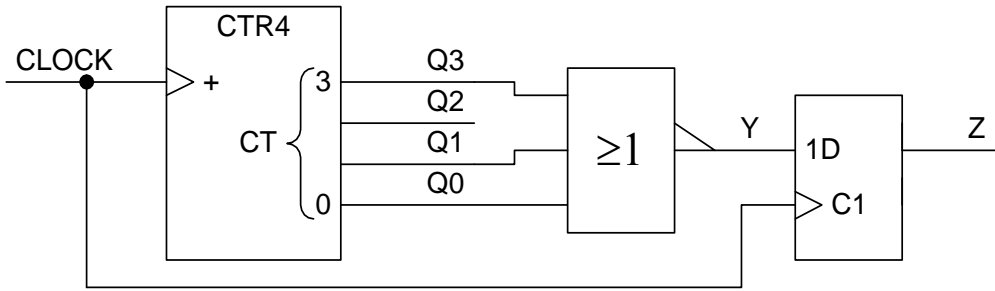
•Y is high when Q=0000 or 0100

- Transition 1 → 2: Q=00?? which includes 0000
- Transition 5 → 6: Q=01?? which includes 0100
- Transition 7 → 8: Q=???? which includes both

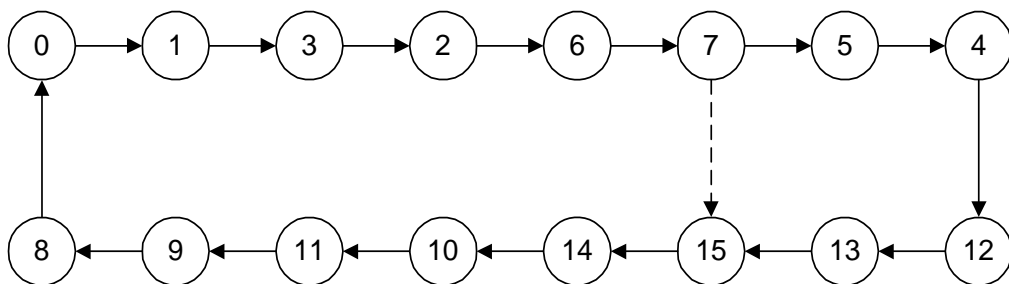


## Eliminating Output Glitches

We can eliminate output glitches by delaying Y with a flipflop:



Alternatively use a count sequence where only one bit changes at a time (e.g. Gray code):



Top and bottom rows differ only in the MSB  $\Rightarrow$  any even count length can be made by branching to the bottom row after half the counts. Dashed line gives a  $\div 12$  counter.

## Quiz Questions

1. If the CLOCK period is  $T$ , what is the range of possible time delays between a change in the DATA input of a shift register and the resultant change in the output of the first stage?
2. How do you combine the outputs of a shift register to generate a pulse for both the rising and the falling edges of its input signal?
3. In order to guarantee that a shift register will notice a pulse on its DATA input, how long must a pulse last?
4. If an AND gate is used to combine 2 of the outputs from a 4-bit counter, how many different count values will make the AND gate output go high?
5. Why do output glitches not occur when a counter counts from 6 to 7?
6. Name two ways in which output glitches may be avoided.

Answers are all in the notes.

## Lecture 7

### **Data Decoding with a Counter**

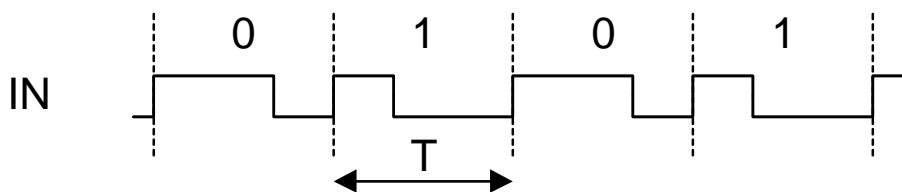
This design example illustrates

- Using a counter to measure time intervals
- The logic symbol notation for a bidirectional counter
- Why it is necessary to use a flipflop to synchronise an asynchronous input signal
- Detailed timing analysis for asynchronous signals
- Assembling a larger design bit by bit

## Data Decoding

**Task:** Decode a data stream where a 0 or 1 is transmitted as a pulse lasting  $\frac{2}{3}T$  or  $\frac{1}{3}T$  respectively.

Problem: you don't know the value of  $T$ .



**Method:**

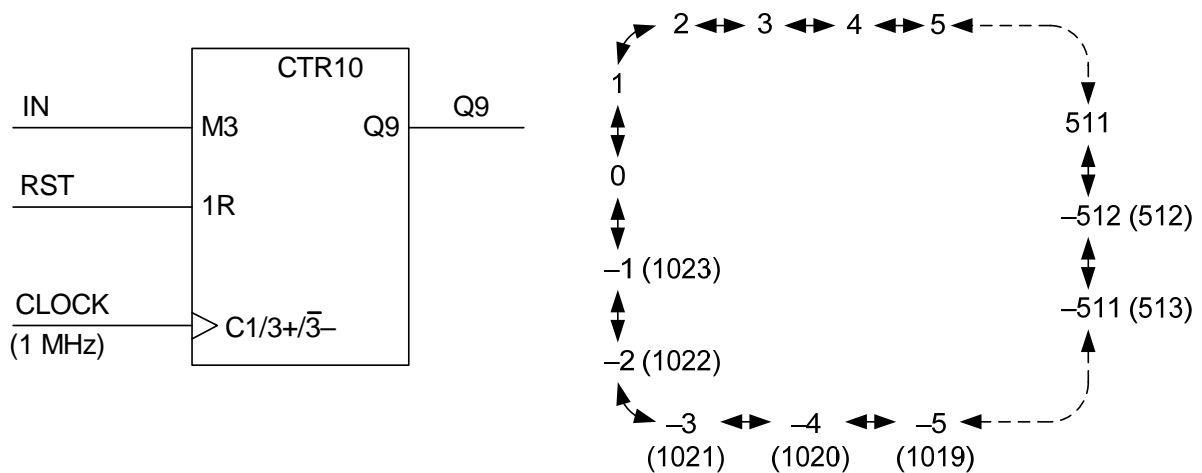
- Wait for a rising edge
- Time how long until the next falling edge
- Time how long until the next rising edge
- Output a 0 or 1 according to which is longer and then go back to (b).

### How do you measure time intervals ?

With a counter.

- Reset the counter at the rising edge
- Count upwards while  $IN=1$
- Count downwards while  $IN=0$
- See if it is +ve or -ve just before you reset it at the next rising edge.

## Counter Symbol



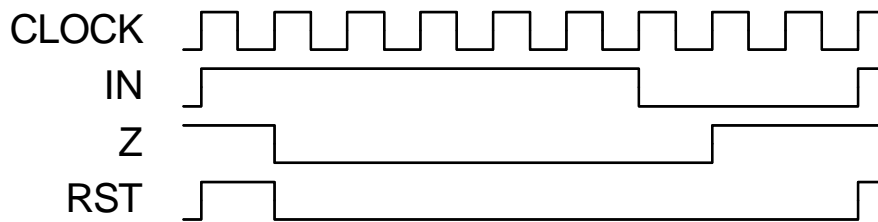
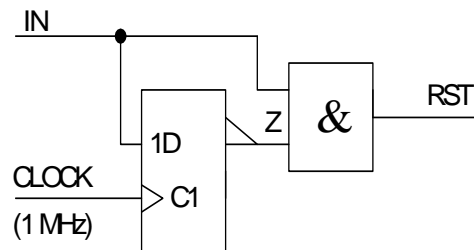
### Notation:

- **M3** is a “mode” input which controls the counting direction. We connect this to IN.
- The **CLOCK** input is
  - +ve edge triggered – indicated by the “>” symbol
  - Has three separate functions divided by “/”
    - C1 means it is a clock for some other feature of the circuit
    - 3+ means that the counter increments on each CLOCK rising edge if the M3 input is high
    - 3- means that the counter decrements on each CLOCK rising edge if the M3 input is low
- **1R**: The “1” means that this input only has any effect when **C1** is active (i.e. the rising edge of CLOCK). **R** means the RST input sets the counter to zero when it is high.
- **CTR10** means it is a 10 bit counter: 0 to 1023. It will wrap around from 1023 to 0 when counting up and from 0 to 1023 when counting down so 1023 is equivalent to -1. Q9, the MSB, tells you when it is negative.

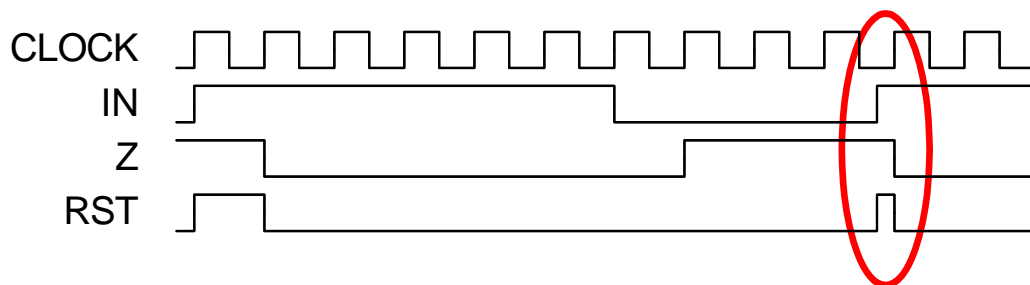
## Resetting the Counter

**Task:** We want to reset the counter on every rising edge of IN.

**Method:** Use a 1-bit shift register to generate a reset pulse.

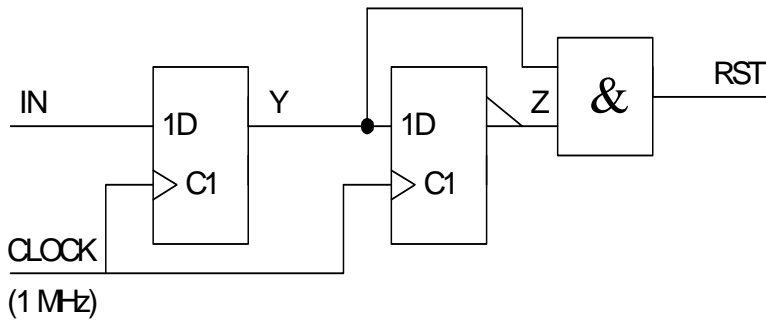


- Z is an inverted version of IN but is 1 clock cycle later.
- RST goes high for one clock cycle every time IN goes high
- **Problem 1:** If IN is unsynchronised (can change at any part of the CLOCK cycle), we might get very short RST pulses.



## Getting Rid of Glitches

**Solution 1:** synchronise IN. Y is always synchronised below.

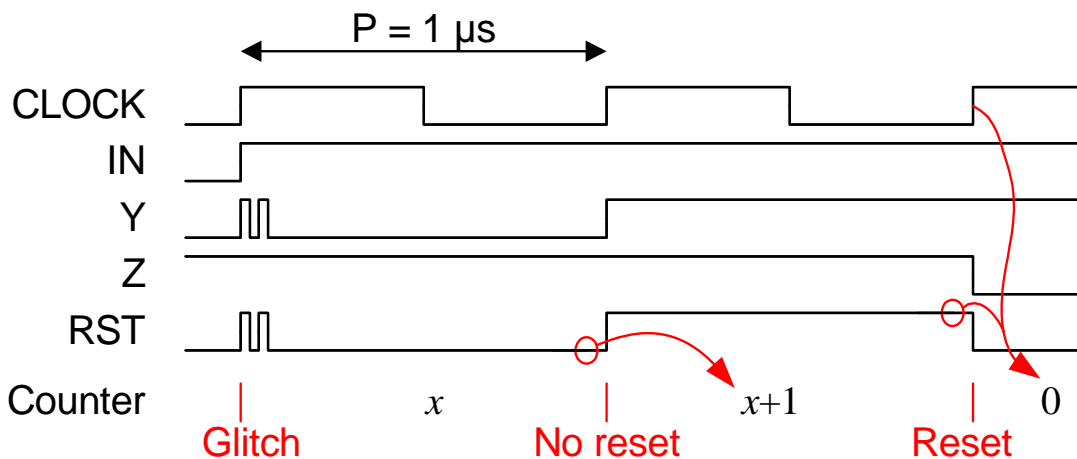


All changes of Y occur just after the clock rising edge.

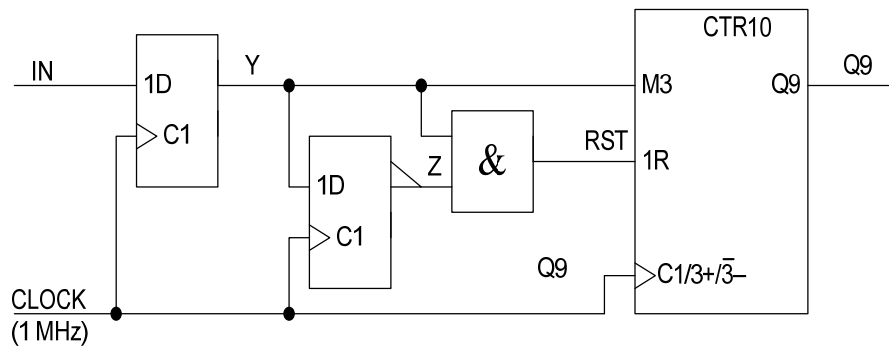
### Potential Problem 2:

If IN changes just on the clock edge, Y (and RST) could oscillate.

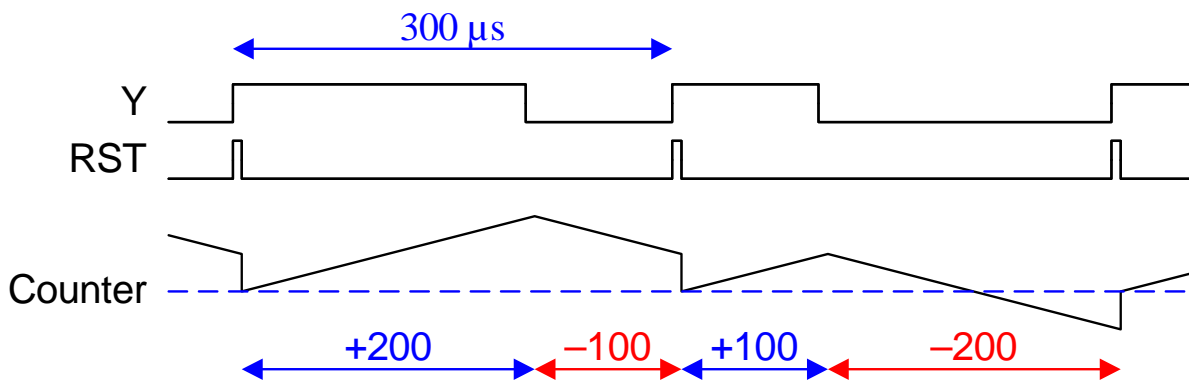
Doesn't matter because the counter only looks at RST on the next clock rising edge and the oscillation will be gone by then.



## Timing the input pulses



- Count **up** when Y is high and down when it is **low**
- Each bitcell lasts  $300 \mu\text{s} \Rightarrow 300$  clock cycles



For a logic **zero**

- Count up by 200 then down by 100  $\Rightarrow +100$  at end of cell

For a logic **one**

- Count up by 100 then down by 200  $\Rightarrow -100$  at end of cell

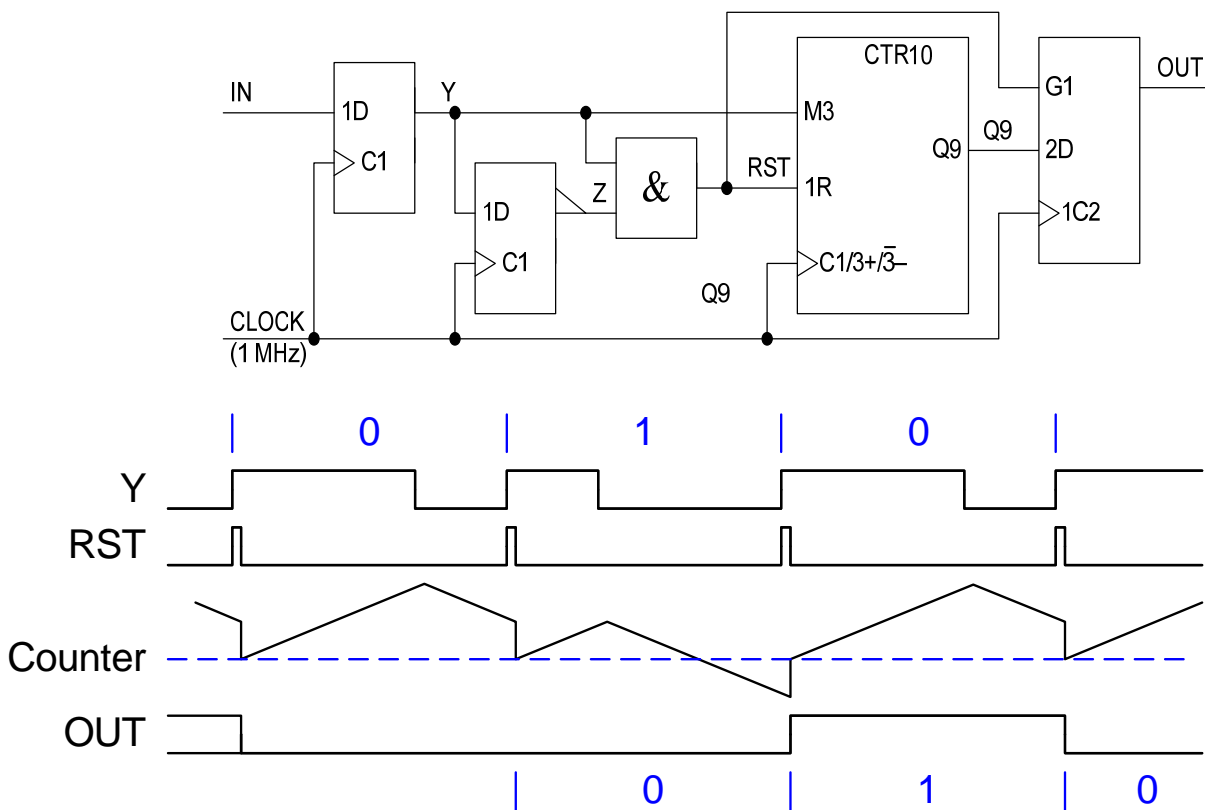
**Counter MSB, Q9, is 0 for positive numbers and 1 for negative**



### Saving the Answer

We need to remember the value of Q9 just before the counter is reset.

- Use the RST pulse to enable the clock of a flipflop
  - G1 is a “gating” input: it enables something when it is high
  - 1C2 is a clock input but only when G1 is true



OUT gives the decoded data stream but one bitcell late.

### Slowest and Fastest Data Rate

Clock =  $1/P$  Hz, Bitcell =  $T$  seconds, Counter =  $n$  bits

#### Slowest Data Rate

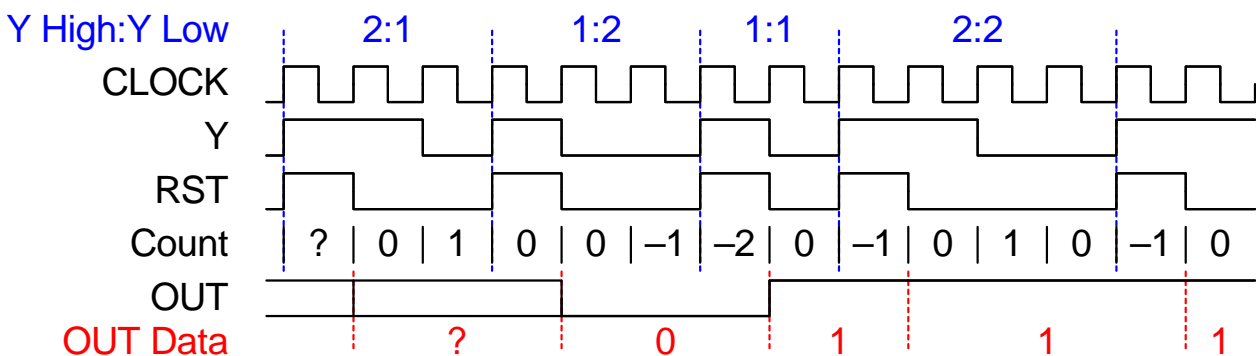
At the end of the bitcell, counter reaches  $\pm T/3P$ . To ensure that Q9 is correct, this must not exceed half the counter range. Hence

$$T/3P < 0.5 \times 2^n \Rightarrow T < 1.5 \times 2^n \times P = 1536 \mu s$$

It doesn't matter if the counter exceeds this range in the middle of a cell: only the final value matters.

#### Fastest Data Rate

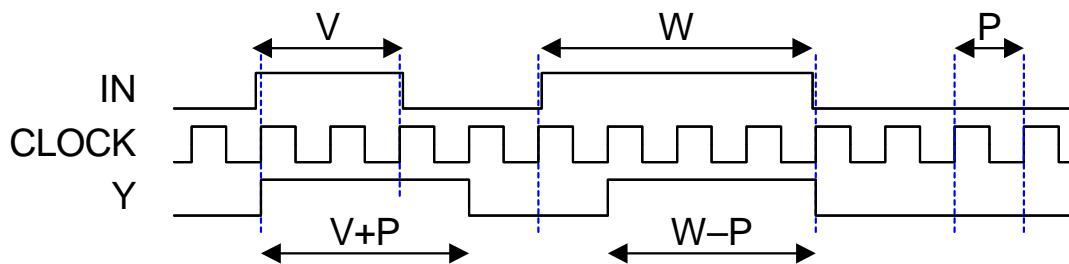
OUT only goes low if Y goes high for **more** cycles than low.



We have to make sure that when IN high:low =  $2/3 T : 1/3 T$  this results in Y being high for more clock cycles than it is low.

### Fastest Data Rate

If a pulse on IN has length  $W$  then the length of the corresponding synchronised pulse on Y is  $W \pm P$ .



If IN high:low =  $\frac{2}{3}T : \frac{1}{3}T$

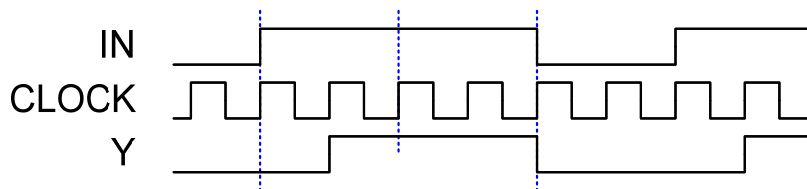
then Y high:low =  $\frac{2}{3}T \pm P : \frac{1}{3}T \pm P$

it follows that we need

$$\frac{2}{3}T \pm P > \frac{1}{3}T \pm P \Rightarrow \frac{2}{3}T - P > \frac{1}{3}T + P$$

$$\Rightarrow 2T - 3P > T + 3P \Rightarrow T > 6P = 6 \mu\text{s}$$

### Example of failure when $T = 6 \mu\text{s}$



If **rising** edges of IN are just too late to be sensed by the clock but **falling** edge is just early enough then Y is high for 3 cycles and low for three cycles  $\Rightarrow$

## Quiz Questions

1. If a flipflop input is labelled “2C1” what is its function ?
2. If a counter input is labelled “ $C1/3+/\bar{3}-$  ” what is its function?
3. What is the difference between a synchronous and an asynchronous reset input to a counter ?
4. Why doesn't it matter if the input to an asynchronous reset input has glitches just after the clock rising edge?
5. If a 10-bit counter initially contains 1020 and is then incremented 10 times, what value will it then contain?
6. What is the minimum and maximum number of clock rising edges included in an asynchronous pulse that lasts  $x$  clock cycles?
7. What is the smallest values of  $x$  to guarantee that
  1.  $\text{ceil}(x) \leq \text{floor}(2x)$
  2.  $\text{ceil}(x) < \text{floor}(2x)$

Answers are all in the notes.

## Lecture 8

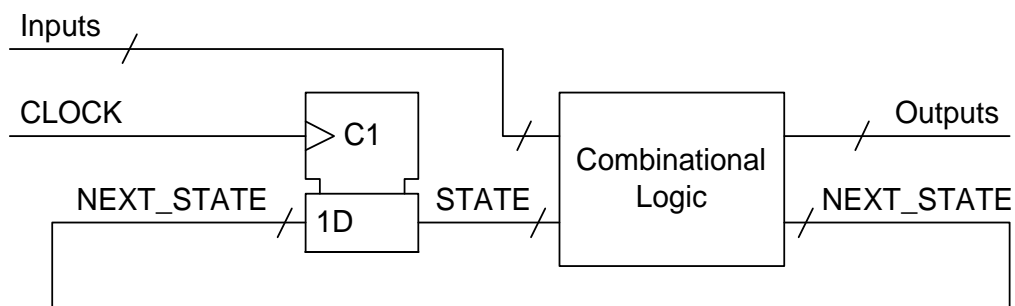
# Synchronous State Machine Analysis

### Objectives

- Review the definition of a synchronous state machine
- Learn how to construct the state table and state diagram of a state machine from its circuit diagram
- Appreciate the alternative ways of drawing the state diagram
- Learn how to draw the output waveforms of a state machine given its initial state and input waveforms
- Understand the causes of glitches in state machine outputs

## Synchronous State Machines

Synchronous State Machine = Register + Logic

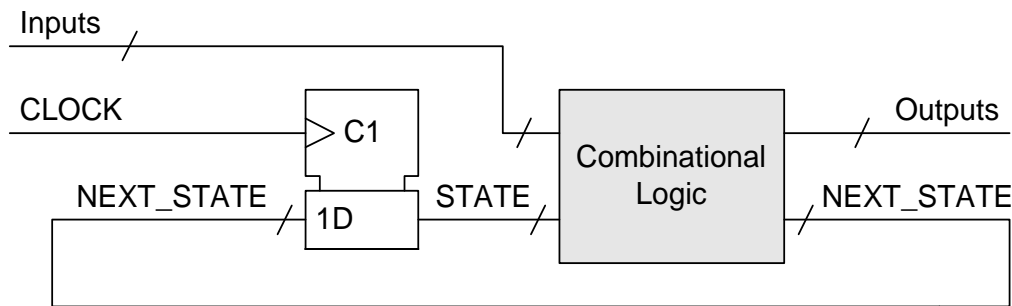


- The *state* is defined by the register contents
- Register has  $n$  flipflops  $\Rightarrow 2^n$  states
- The state only ever changes on  $\text{CLOCK}\uparrow$ 
  - We stay in a state for an exact number of  $\text{CLOCK}$  cycles
- The state is the only memory of the past

### Rules:

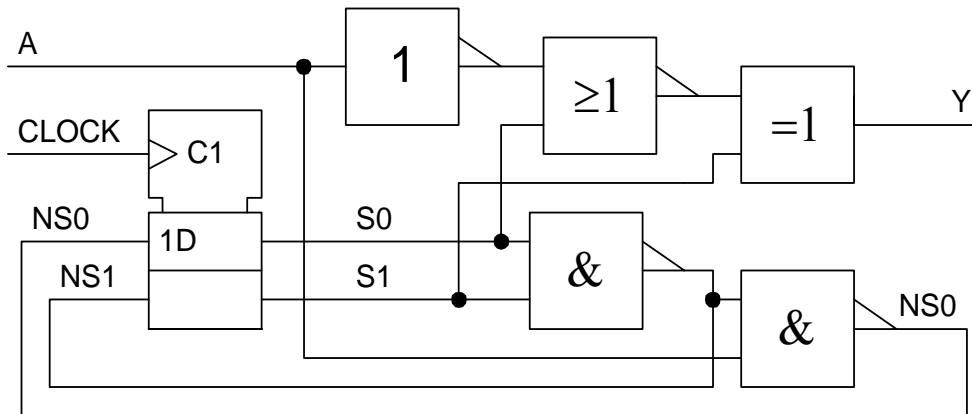
- Never mess around with the clock signal
- Never use *asynchronous* SET/RESET inputs to register (*asynchronous* = independent of  $\text{CLOCK}$ )

## Combinational Logic Block



- The combinational logic outputs specify two things:
  - ★ **The output signals during the current state**  
These may change during the state if the inputs change
  - ★ **Which state to go to at the next CLOCK ↑**  
This too may change during a state but the only thing that matters is its value just before CLOCK ↑
- *combinational* logic has no internal feedback loops  
⇒ no memory
  - combinational logic outputs are entirely determined by the **current STATE** and the **current Inputs**

## Analysing a State Machine



### State Table:

Truth table for the combinational logic:

- One row per state:  $n$  flipflops  $\Rightarrow 2^n$  rows
- One column per input combination:  
 $m$  input signals  $\Rightarrow 2^m$  columns
- Each cell specifies the *next state* and the *output signals during the current state*
  - for clarity, we separate the two using a /

NS1,NS0/Y		
S1,S0	A=0	A=1
00	11/0	10/1
01	11/0	10/0
10	11/1	10/0
11	01/1	01/1



## Drawing the State Diagram

Split state table into two parts:

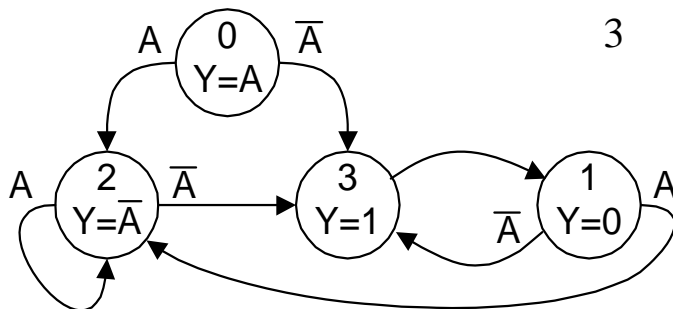
NS1,NS0/Y		
S1,S0	A=0	A=1
00	11/0	10/1
01	11/0	10/0
10	11/1	10/0
11	01/1	01/1

Next State: NS1:0

S1:0	A=0	A=1
0	3	2
1	3	2
2	3	2
3	1	1

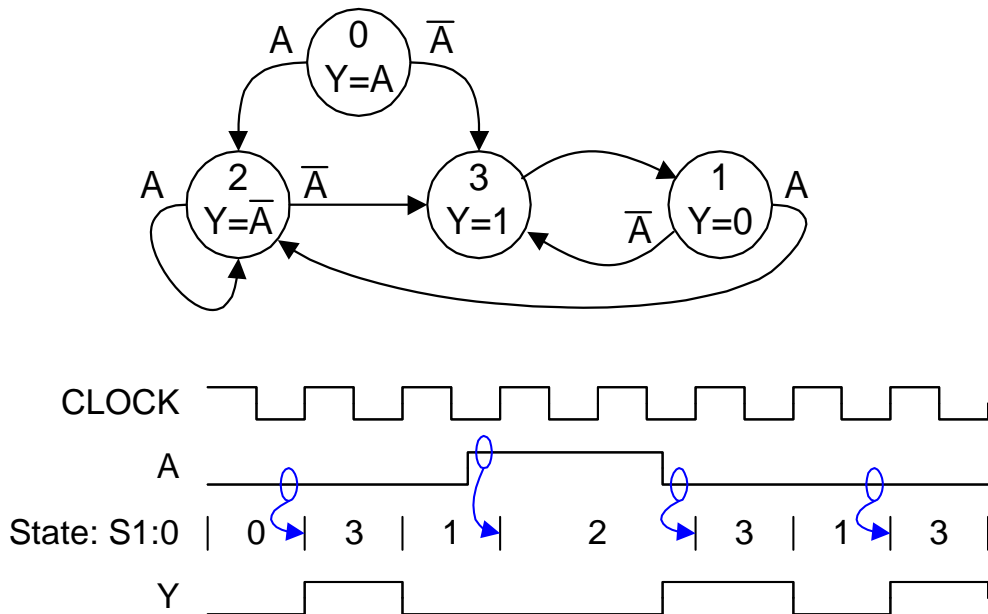
Output Signal: /Y

S1:0	A=0	A=1	Y
0	/0	/1	Y=A
1	/0	/0	Y=0
2	/1	/0	Y=!A
3	/1	/1	Y=1



- Transition arrows are marked with Boolean expressions saying when they occur
  - Every input combination has exactly one destination.
  - Unlabelled arrows denote unconditional transitions
- Output Signals: Boolean expressions within each state.

## Timing Diagram



State machine behaviour is entirely determined by:

- The initial state
- The input signal waveforms

### State Sequence:

*Determine this first.* Next state depends on input values just before CLOCK ↑.

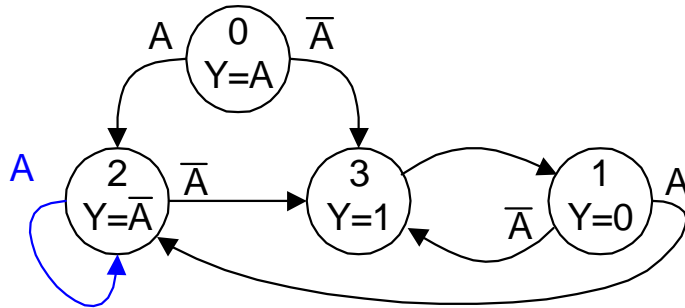
### Output Signals:

Defined by Boolean expressions within each state.

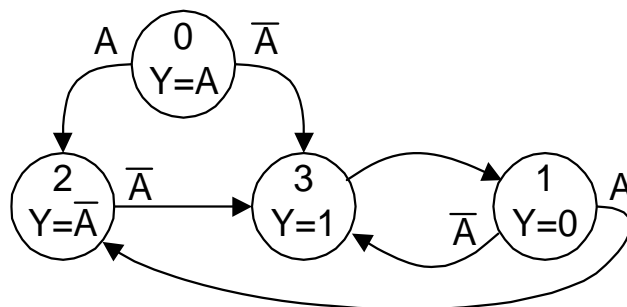
If all the expressions are constant 0 or 1 then outputs only ever change on clock ↑. (*Moore machine*)

If any expressions involve the inputs (e.g. Y=A) then it is possible for the outputs to change in the middle of a state. (*Mealy machine*)

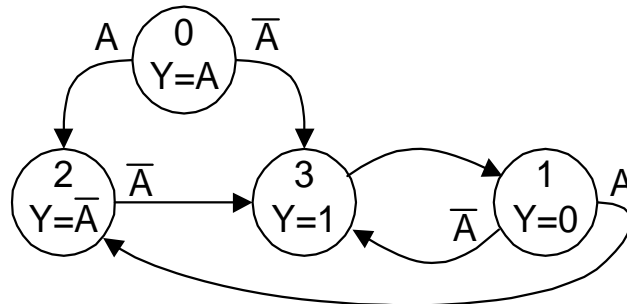
## Self-Transitions



- We can omit transitions from a state to itself.
  - Aim: to save clutter on the diagram.
- The state machine remains in its current state if none of the transition-arrow conditions are satisfied.
  - From state 2, we go to state 3 if !A occurs, otherwise we remain in state 2.

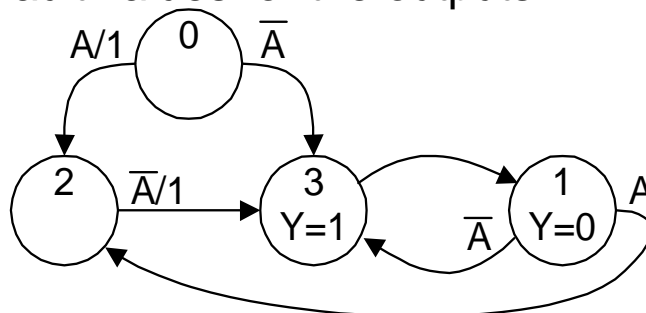


## Output Expressions on Arrows



It may make the diagram clearer to put output expressions on the arrows instead of within the state circles:

- Useful if the same Boolean expression determines both the *next state* and the *output signals*.
- For each state, the output specification must be *either* inside the circle *or else* on *every* emitted arrow
- If self transitions are omitted, we must declare default values for the outputs



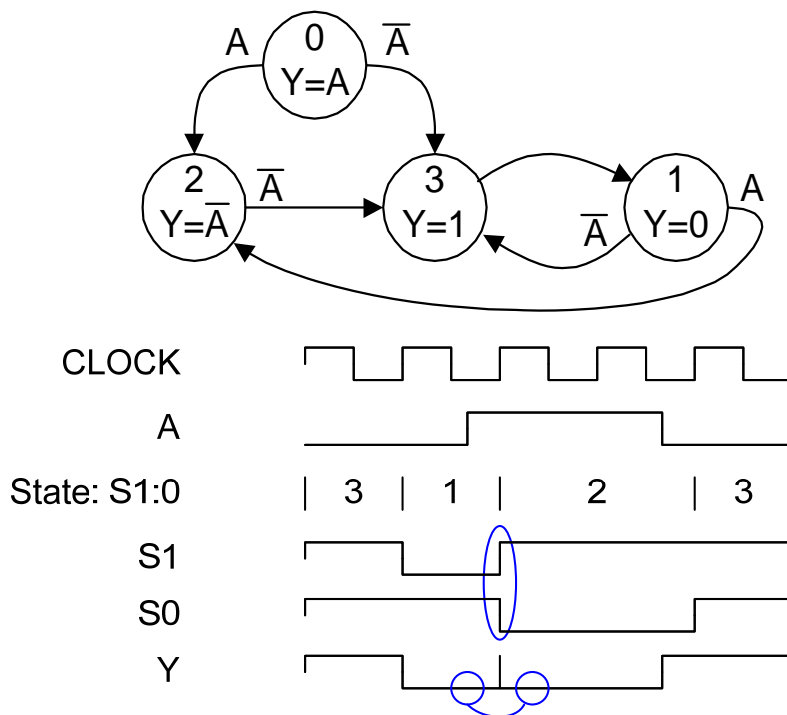
Output: /Y  
Default: Y=0

- Outputs written on an arrow apply to the state *emitting* the arrow.
- Outputs still apply for the entire time spent in a state
- This does not affect the Moore/Mealy distinction
- This is a notation change only

## Output Glitches

When making a transition from one state to another, the logic is likely to generate a glitch on an output if:

- two or more state bits change
- the output has the same value in both states
- some combination of the changing state bits would cause the output to change

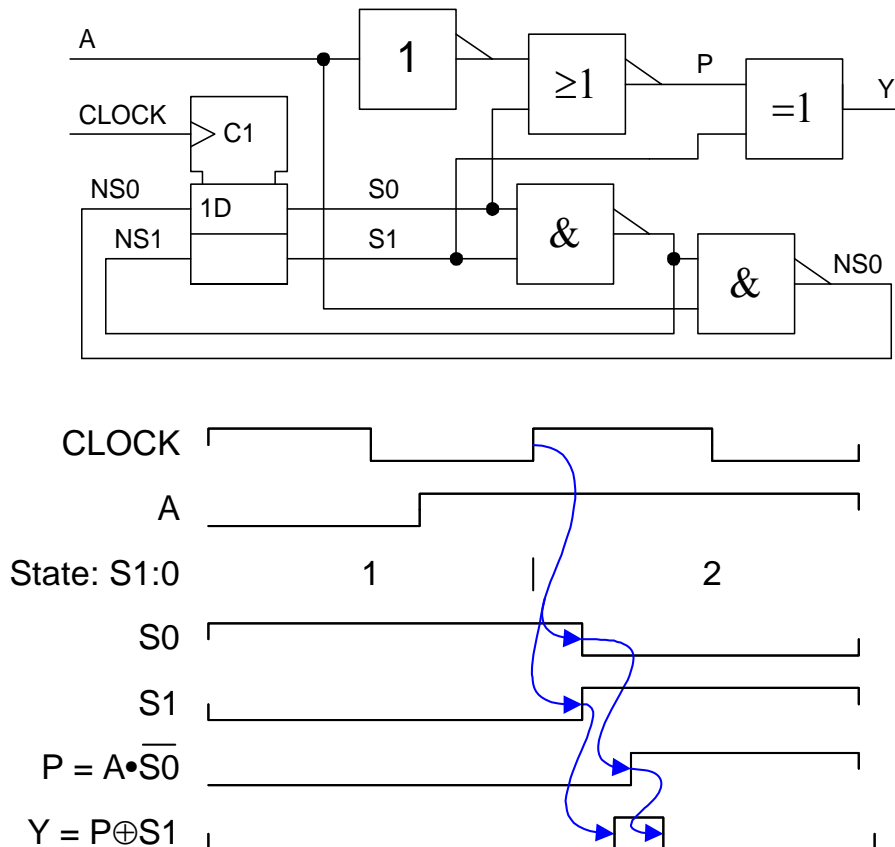


In changing from state 1 to state 2:

- the two states differ in both S0 and S1
- the output is low in both states
- if S0 and S1 both went high then the output would change.

## Cause of Output Glitches

Look in detail at the logic when going from state 1 to 2:



The two inputs to the XOR gate (P and S1) are meant to change simultaneously.

In fact S1 changes first because of the delay through the NOR gate.

The XOR gate “sees” the effect of S1 changing before it “sees” the effect of S0 changing. It is as if we went briefly into state 3.

## Quiz Questions

1. What is the definition of a Moore machine?
2. What does it mean if an arrow in a state diagram has no Boolean expression attached to it?
3. To which state does an output value refer when it is marked on an arrow in a state diagram? Is it the state the arrow points *towards* or the state the arrow points *away from*?
4. Is the next state determined by the value that the input signals have just *before* or just *after* the CLOCK↑?
5. If transitions from a state to itself have been omitted from a state diagram, how can you tell when such a transition occurs?
6. What are the three conditions that give rise to output glitches?

Answers are all in the notes.

## Lecture 9

# Synchronous State Machine Design

### Objectives

- To learn how to design a state machine to meet specific objectives
- To understand when two or more states are equivalent and can be merged into a single state.
- To understand the principles of assigning state numbers
- To appreciate when it is necessary to synchronise a state machine's inputs with the CLOCK
- To understand how a state machine is implemented using programmable logic



## Designing a Synchronous State Machine

The state is the only way the circuit can remember what happened in the past.

The number of states required equals the number of past histories that the circuit needs to distinguish.

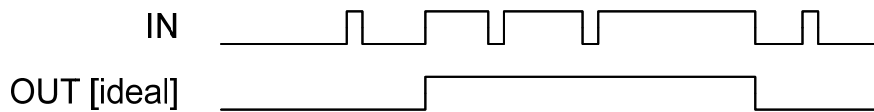
### General Design Procedure

- Construct a sequence of input waveforms that includes all relevant situations.
- Go through the sequence from the beginning. Each time an input changes, you must decide:
  - branch back to a previous state if the current situation is materially identical to a previous one
  - create a new state otherwise
- For each state you must ensure that you have specified:
  - which state to branch to for every possible input pattern
  - what signals to output for every possible input pattern

## Designing a Noise Pulse Eliminator

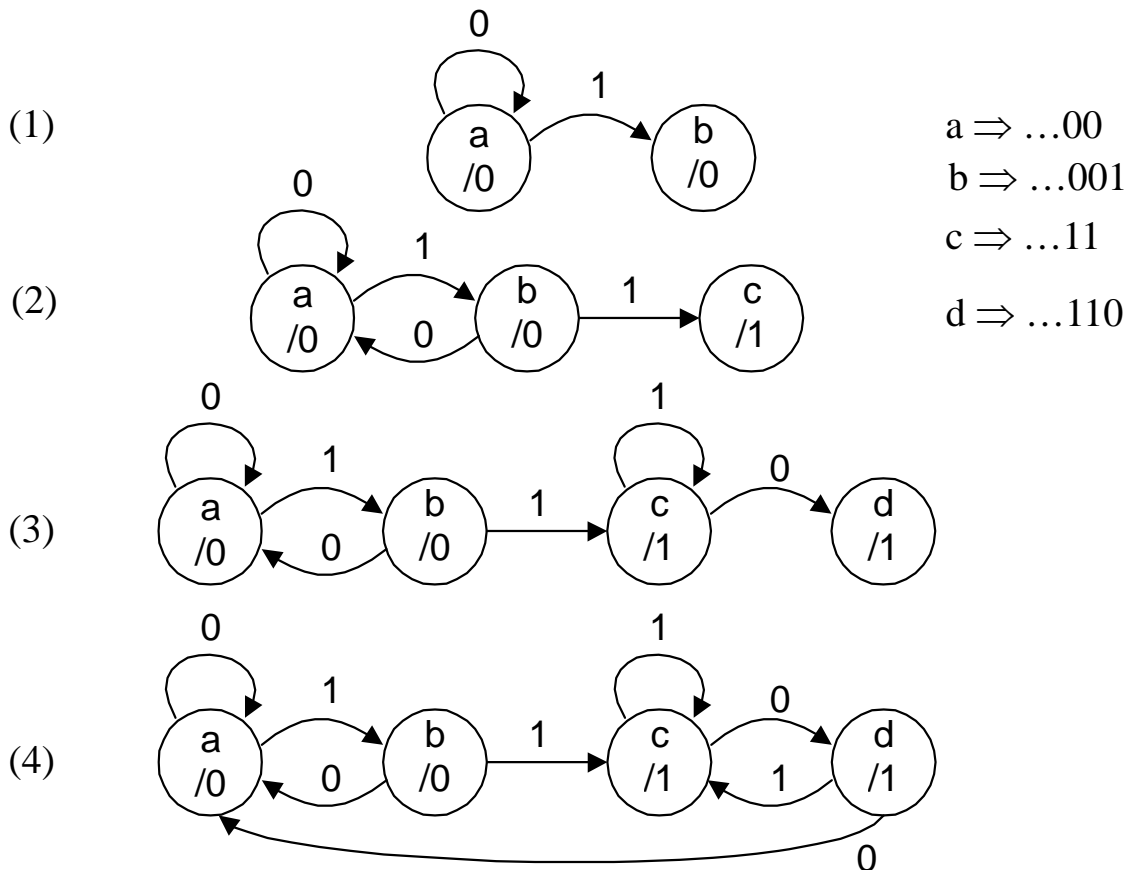
### Design Problem: Noise elimination circuit

- We want to remove pulses that last only one clock cycle



- Use letters a,b,... to label states; we choose numbers later.
- Decide what action to take in each state for each of the possible input conditions.
- Use a Moore machine (i.e. output is constant in each state).  
Easier to design but needs more states & adds output delay.

Assume initially in state “a” and IN has been low for ages



## Explanatory Notes

- (1) If IN goes high for two (or more) clock cycles then OUT must go high, whereas if it goes high for only one clock cycle then OUT stays low. It follows that the two histories “IN low for ages” and “IN low for ages then high for one clock” are different because if IN is high for the next clock we need different outputs. Hence we need to introduce state b.
- (2) If IN goes high for one clock and then goes low again, we can forget it ever changed at all. This glitch on IN will not affect any of our future actions and so we can just return to state a. If on the other hand we are in state b and IN stays high for a second clock cycle, then the output must change. It follows that we need a new state, c.
- (3) The need for state d is exactly the same as for state b earlier. We reach state d at the end of an output pulse when IN has returned low for one clock cycle. We don't change OUT yet because it might be a false alarm.
- (4) If we are in state d and IN remains low for a second clock cycle, then it really is the end of the pulse and OUT must go low. We can forget the pulse ever existed and just return to state a.

**Each state represents a particular history that we need to distinguish from the others:**

- |                       |                      |
|-----------------------|----------------------|
| (a) IN=0 for >1 clock | (b) IN=1 for 1 clock |
| (c) IN=1 for >1 clock | (d) IN=0 for 1 clock |

## Equivalent States

An initial design often creates more states than are necessary.

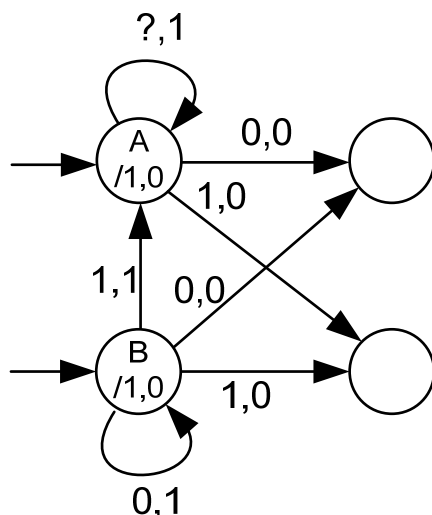
**States A and B are said to be equivalent if, for any possible input sequence, you get identical output waveforms regardless of whether the initial state is A or B.**

You can simplify a state machine by merging equivalent states into a single state.

Two states are definitely equivalent if:

- They have the same outputs for every possible input combination.
- They have the same next state for every possible input combination (assuming they themselves are equivalent).

This rule won't always find all possible equivalent states and so won't necessarily make the state machine as simple as possible (you will learn a complete rule next year).



States A and B are equivalent

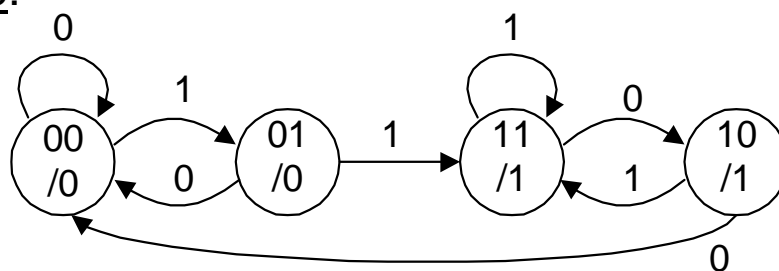
## Implementing a State Machine

Assign each state a unique binary number. Your choice affects circuit complexity but the circuit will work correctly whatever choice you make.

### State Assignment Guidelines:

- Any outputs that depend only on the state should if possible be used as some of the state bits.
- Assign similar (=most bits the same) numbers to states (a) that are linked by arrows, (b) that share a common destination or source, (c) that have the same outputs.
- If two subsets of the state diagram have identical transitions with identical input conditions, they should be numbered so that corresponding states have similar numbers.

### Example:



State Numbers: S1,S0  
Inputs/Outputs: IN/OUT

- S1 is the same as OUT (from the first guideline)
- All states linked by arrows differ in only one bit (from the second guideline)

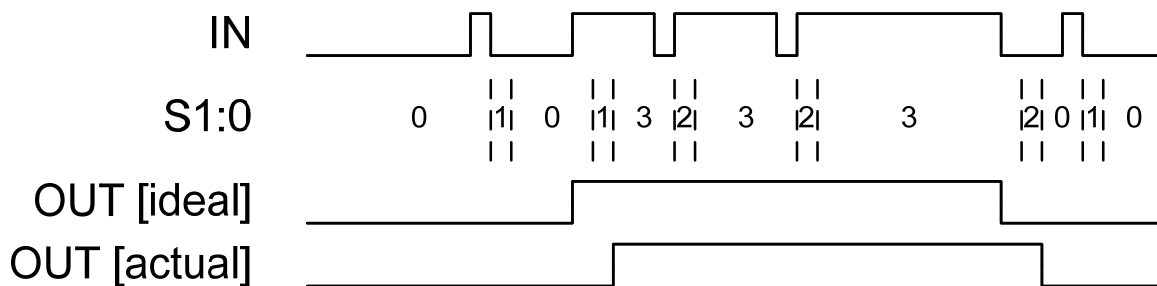
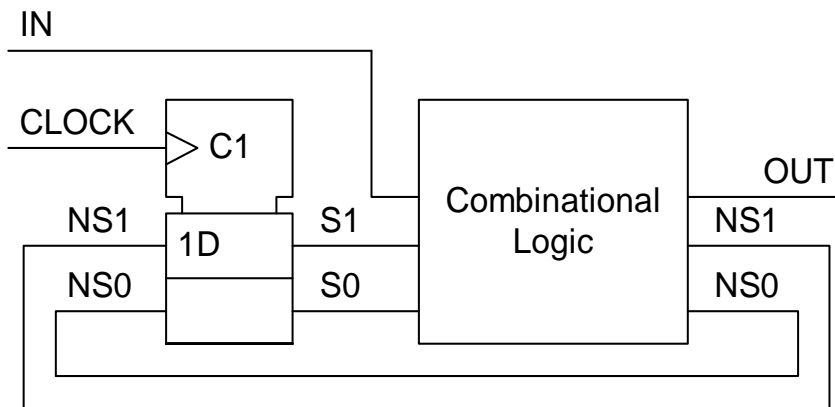
## Implementing a State Machine (contd)

Now we can draw a Karnaugh map (really three K-maps in one) giving NS1, NS0 and OUT in terms of S1, S0 and IN:

		NS1,NS0/OUT	
S1,S0		IN=0	IN=1
00		00/0	01/0
01		00/0	11/0
11		10/1	11/1
10		00/1	11/1

From this we can derive Boolean expressions for the combinational logic block:

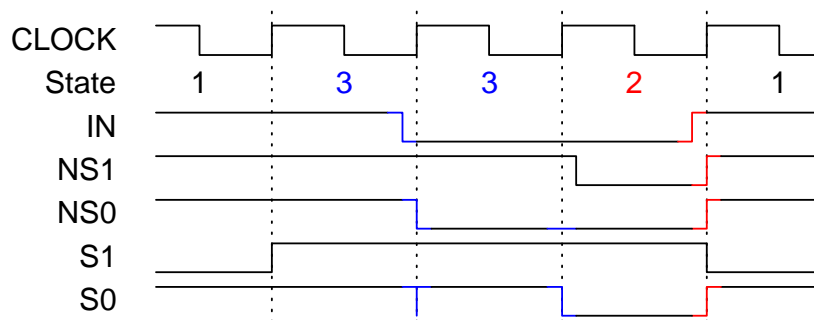
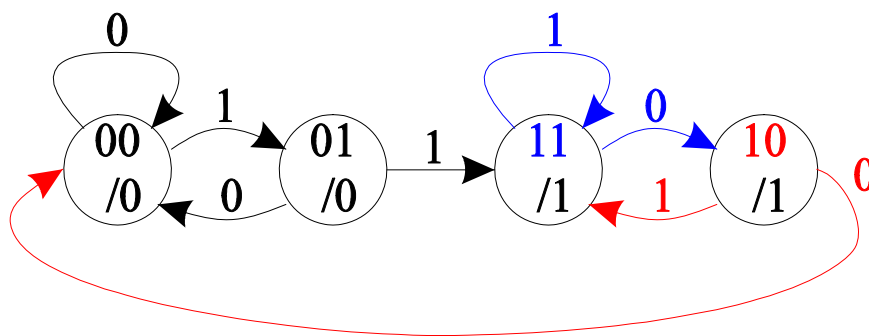
$$NS1 = IN \cdot (S1 + S0) + S1 \cdot S0 \quad NS0 = IN \quad OUT = S1$$



## Unsynchronised Inputs

An input transition just before CLOCK ↑ can cause the NS bits to change within the setup/hold window of the register.

If  $k$  of the NS bits change we might go to any of  $2^k$  states:



### State 3:

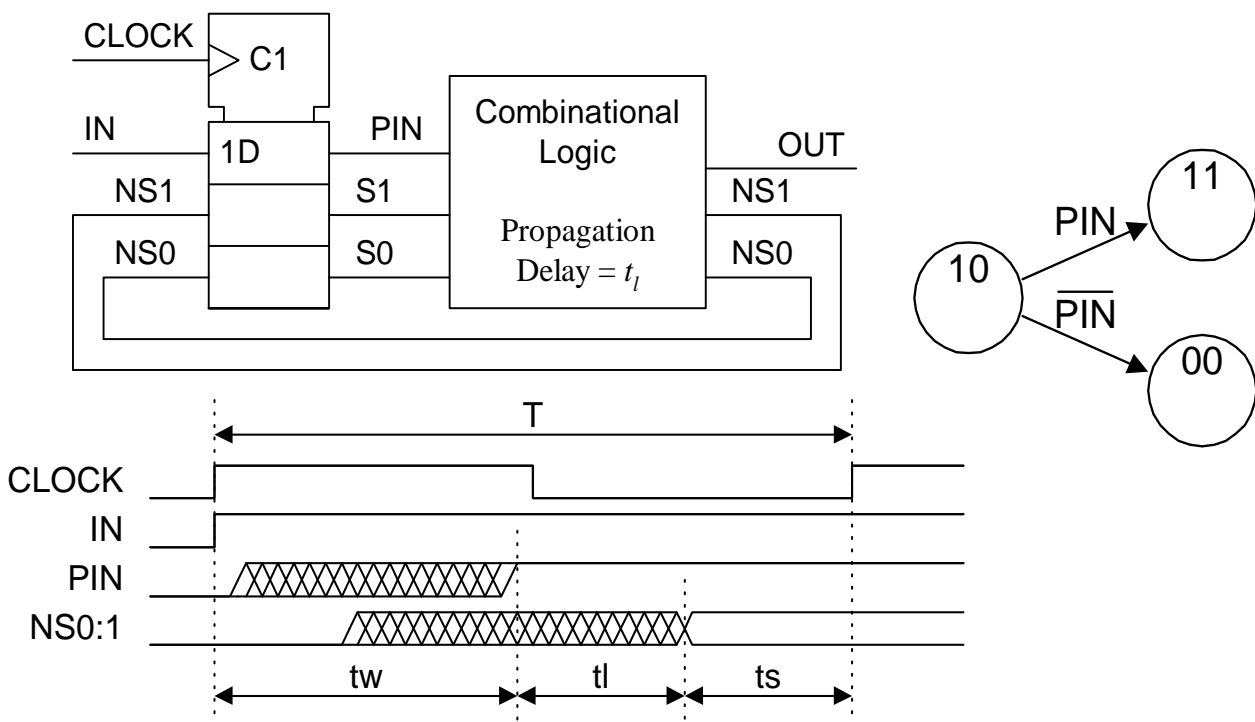
IN ↓ causes NS0:1 to change from 11 to 10 ⇒  $k=1$ .  
 NS0 ↓ too late for S0 but causes glitch on S0  
 S0 goes low on next CLOCK ↑. Everything is OK.

### State 2:

IN ↑ causes NS0:1 to change from 00 to 11 ⇒  $k=2$ .  
 NS0 ↑ changes in time so S0 → 1.  
 NS1 ↑ changes too late so S1 → 0.  
 Next state is 01 which is an ILLEGAL destination.

## Input Synchronization

- An asynchronous input must be synchronized if in any state it affects more than one of the next state bits.
- Inputs can be synchronized by passing them through a register before they go to the combinational logic:



- Here IN must be synchronized because destinations 11 and 00 differ in more than 1 bit position
- IN might change within setup-hold window
- PIN (Previous IN) will be stable  $t_w$  after CLOCK  $\uparrow$   
Typical  $t_w$  is 25ns for MTBF of 1000 years
- NS1:0 will be stable  $t_w + t_l$  after CLOCK  $\uparrow$
- CLOCK period ( $T$ ) must be greater than  $t_w + t_l + t_s$  for reliable operation
- To get a huge MTBF, send PIN through a 2nd register

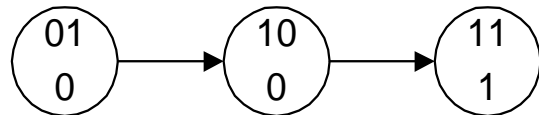


## Input Sync versus Output Glitches

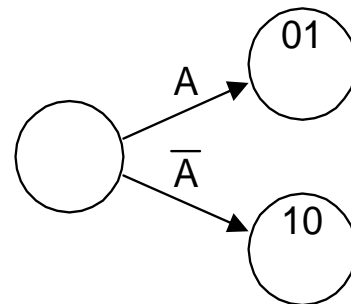
Do not confuse two different problems:

Output glitches are likely if three conditions are true:

- *two consecutive states* differ in more than one bit position
- output is the same in both states
- changing only some of the state bits would cause an output change



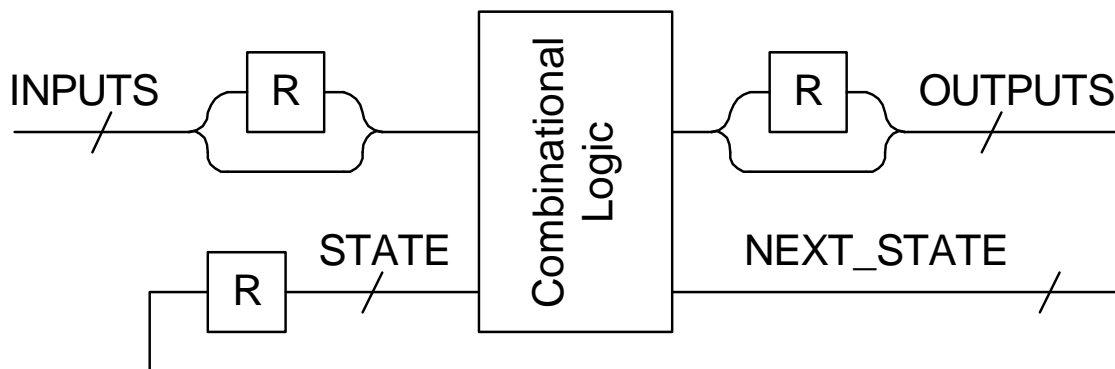
Input synchronisation is needed when *two alternative destinations* differ in more than one bit position.



***This is a far more serious problem as it results in the wrong state sequence.***

In both cases the solution is to send the offending input or output signal through a register/flipflop. (This adds a 1-cycle delay).

## Universal State Machine Circuit Diagram



- “R” denotes register bits: **all with the same CLOCK**
- **Inputs** can go directly into logic block if they are already synchronized with CLOCK. Others must be passed through a register unless (i) they only affect one bit of the Next\_State and (ii) the logic block is hazard-free.
- Glitch-prone **outputs** must be deglitched if they go to a clock or to an asynchronous set/reset/load input.
  - For some state diagrams it is possible to eliminate output glitches by clever state numbering.
- Input synchronization and output deglitching add circuitry and increase input-to-output delays. Avoid if unnecessary.

## Quiz Questions

1. What problem can arise if two alternative next states differ in more than one bit position?
2. What problem can arise if two consecutive states differ in more than one bit position?
3. What determines the minimum number of states needed by a state machine to solve a particular problem?
4. What aspects of a state machine's operation are affected by the assignment of state numbers?
5. Under what conditions can a group of states be merged into a single state?

Answers are all in the notes.

## Lecture 10

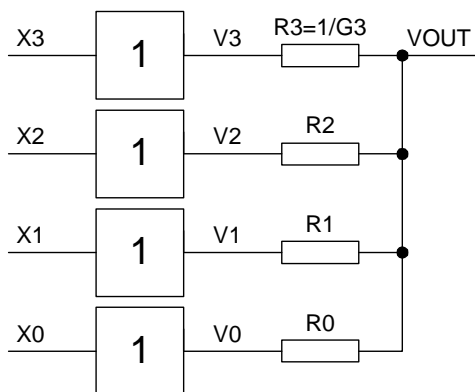
# Digital-to-Analog Conversion

### Objectives

- Understand how a weighted-resister DAC can be used to convert numbers with binary or non-binary bit weightings
- Understand the meaning of the terms used to specify DAC accuracy
- Understand how an R-2R ladder can be used to convert both unsigned and signed binary numbers
- Understand the offset binary representation of negative numbers

## Digital-to-Analog Conversion

We want to convert a binary number into a voltage proportional to its value:



$$(V_3 - V_{OUT})G_3 + \dots + (V_0 - V_{OUT})G_0 = 0$$

$$V_{OUT} = \frac{V_3G_3 + V_2G_2 + V_1G_1 + V_0G_0}{G_3 + G_2 + G_1 + G_0}$$

$$R_{Thevenin} = \frac{1}{G_3 + G_2 + G_1 + G_0}$$

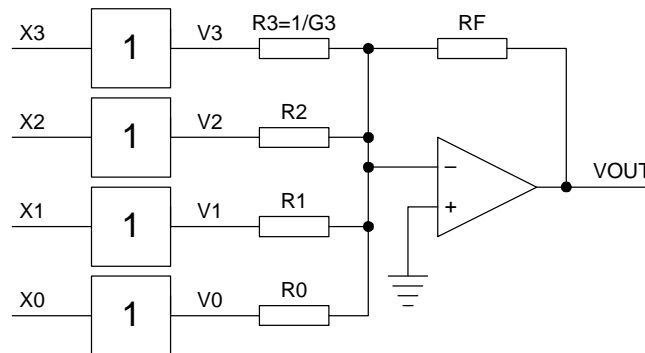
Hence  $V_{OUT}$  is a weighted sum of  $V_3, \dots, V_0$  with weights proportional to the conductances  $G_3, \dots, G_0$ .

- If X3:0 is a binary number we want conductances in the ratio 8:4:2:1.
- Very fast: gate slew rate  $\approx 3$  V/ns.
- We can scale the resistors to give any output impedance we want.

You do not have to use a binary weighting

- By using other conductance ratios we can choose arbitrary output voltages for up to five of the sixteen possible values of X3:0. May need additional resistors from VOUT to the power supplies.

## Output Op-Amp



$$V_{OUT} = \frac{-R_F}{R_{Thévenin}} \times V_{Thévenin} = -R_F (V_3 G_3 + V_2 G_2 + V_1 G_1 + V_0 G_0)$$

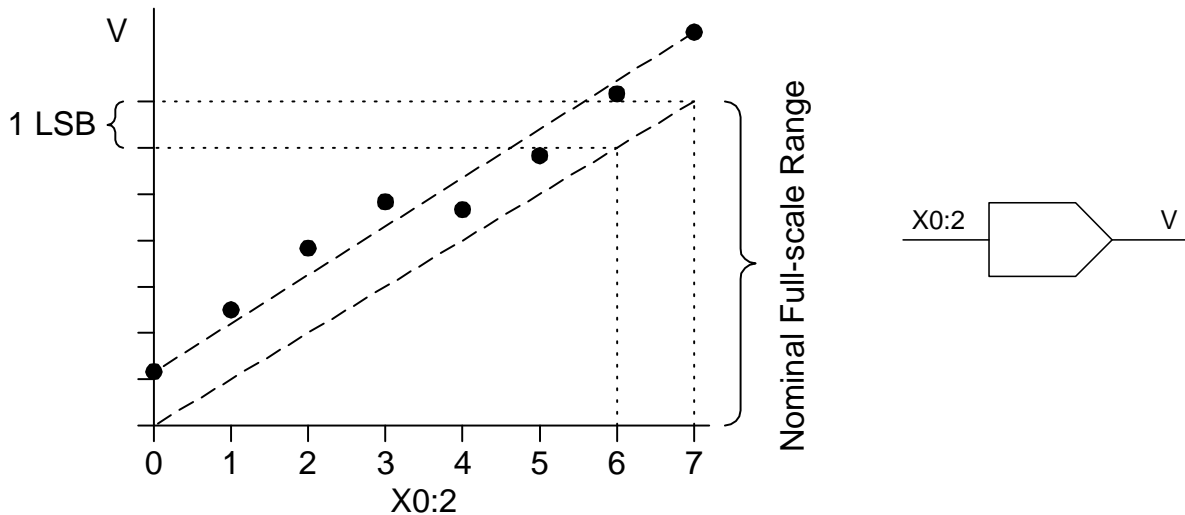
### Adding an op-amp:

- The voltage at the junction of all the resistors is now held constant by the feedback
  - Hence current drawn from  $V_3$  is independent of the other voltages  $V_2, \dots, V_0$
  - Hence any gate non-linearity has no effect  $\Rightarrow$  more accurate.
- Lower output impedance
- Much slower: op-amp slew rate  $\approx 1 \text{ V}/\mu\text{s}$ .

Hard to make accurate resistors covering a wide range of values in an integrated circuit.

- Weighted-resistor DAC is no good for converters with many bits.

## DAC Jargon



Accuracy=1.8@X=3      Linearity=-0.7@X=4  
 Non-monotonic@3→4      Diff Linearity=-1.2@ 3→4  
 (all in units of LSB)

Resolution      1 LSB =  $\Delta V$  when  $X \rightarrow X+1$   
                          = Full-scale range  $\div (2^N - 1)$

Accuracy      Worst deviation from nominal line

Linearity      Worst deviation from line joining end points

Differential Linearity  
 Worst error in  $\Delta V$  when  $X \rightarrow X+1$   
 measures smoothness

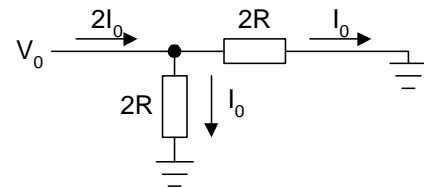
Monotonic      At least  $\Delta V$  always has the correct sign

Settling time      Time taken to reach the final value to within  
 some tolerance, e.g.  $\pm 1/2$  LSB

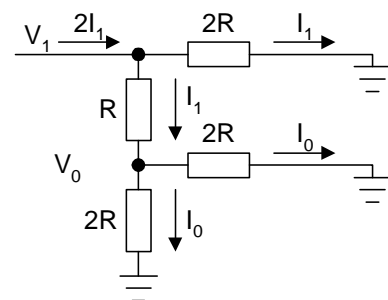
## R-2R Ladder

We want to generate currents  $I_0, 2I_0, 4I_0, \dots$

- Two  $2R$  resistors in parallel means that the  $2I_0$  current will split equally.



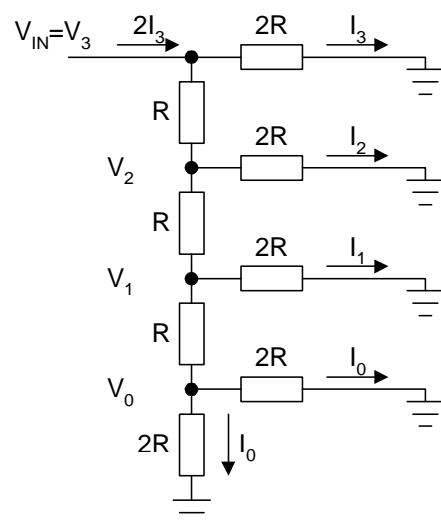
- The Thévenin resistances of the two branches at  $V_1$  both equal  $2R$  so the current into this node will split evenly.



We already know that the current into node  $V_0$  is  $2I_0$ , so it follows that  $I_1=2I_0$ .

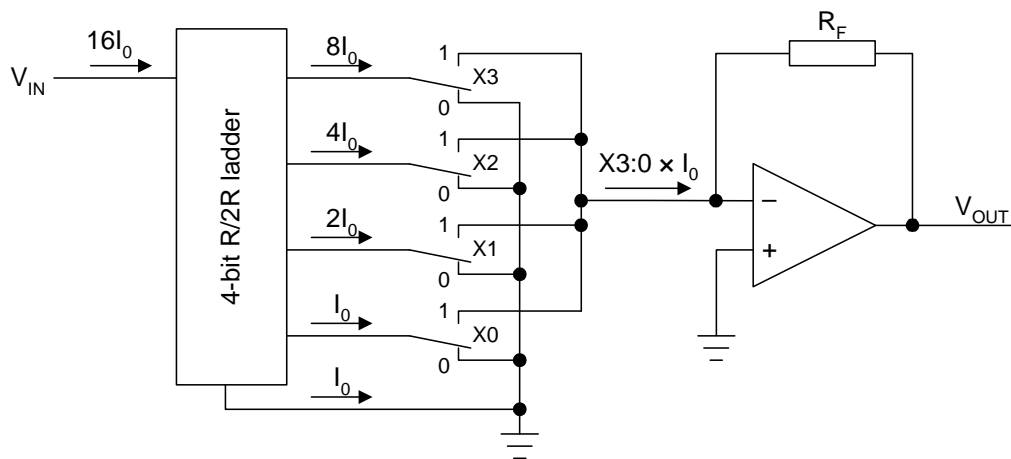
- We can repeat this process indefinitely and, using only two resistor values, can generate a whole series of currents where  $I_n=2^n I_0$ .

From the voltage drop across the horizontal resistors, we see that  $V_n = 2RI_n = 2^{n+1}RI_0$ . For an  $N$ -bit ladder the input voltage is therefore  $V_{in} = 2^N RI_0 \Rightarrow I_0 = 2^{-N} V_{in} / R$ .





## Current-Switched DAC



- Total current into summing junction is  $X_{3:0} \times I_0$   
Hence  $V_{out} = X_{3:0} \times V_{in} / 16R \times -R_f$
- We switch currents rather than voltages so that all nodes in the circuit remain at a constant voltage  
  - $\Rightarrow$  no need to charge/discharge node capacitances
  - $\Rightarrow$  faster.
- Use CMOS transmission gates as switches: adjust ladder resistors to account for switch resistance.
  - Each 2-way switch needs four transistors
- As required by R/2R ladder, all the switch output terminals are at 0 V.
  - ladder outputs are always connected either to ground or to a virtual earth.

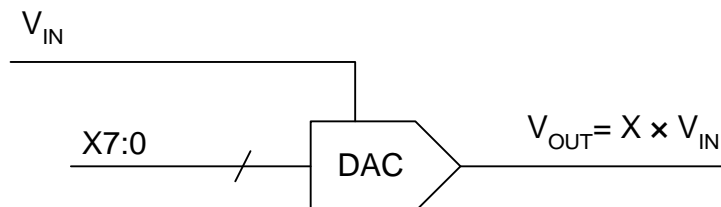
## Digital Attenuator

The output of the DAC is proportional to the *product* of an analog voltage ( $V_{in}$ ) and a digital number ( $X_{3:0}$ ).

$$V_{out} = X_{3:0} \times V_{in} / 16R \times -R_f$$

It is called a *multiplying* DAC.

Can be used as a digital attenuator:



Here the digital number  $X_{7:0}$  controls the gain of the circuit.

## Bipolar DAC

A bipolar DAC is one that can give out both positive and negative voltages according to the sign of its input. There are two aspects of the circuit that we need to change:

### Number Representation

Normally we represent numbers using *2's complement* notation (because we can then use the same addition/subtraction circuits).

For converters it is more convenient to use *offset-binary* notation.

### Positive and Negative Currents

We need to alter our R-2R ladder circuit so that we can get an output current that can be positive or negative according to the sign of the input number.

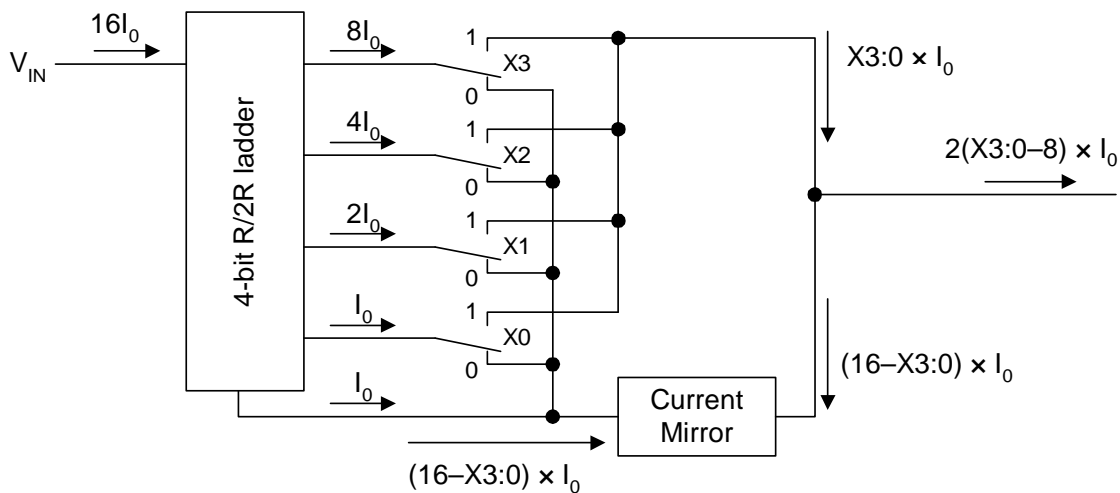
To do this, we will use a *current mirror*.

## Signed Numbers

<u>Value (v)</u>	<u>2's complement (y)</u>	<u>Offset Binary (x)</u>	<u>(u=v+8)</u>
-8	1000	0000	0
-7	1001	0001	1
-6	1010	0010	2
-5	1011	0011	3
...	...	...	
-1	1111	0111	7
0	0000	1000	8
1	0001	1001	9
...	...	...	
6	0110	1110	14
7	0111	1111	15

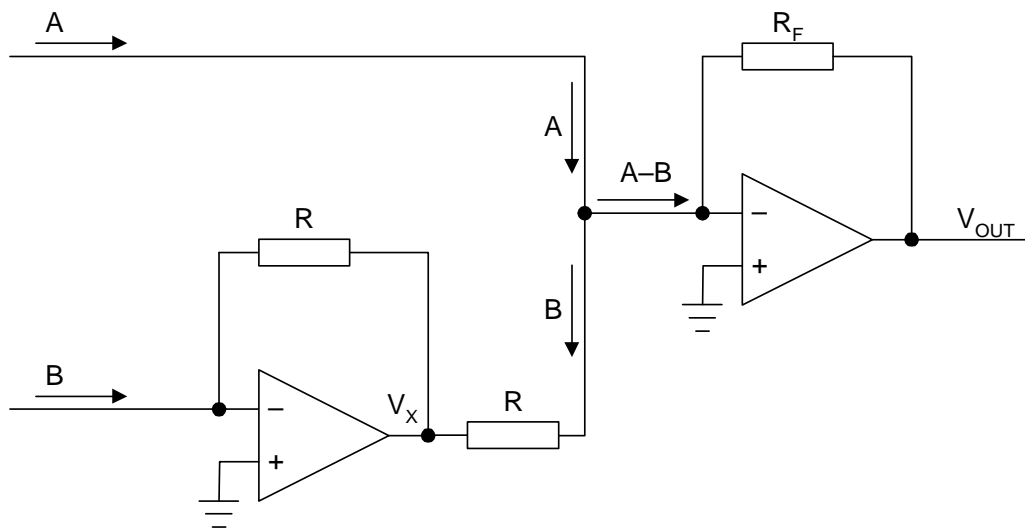
- Obtain offset binary from 2's complement by inverting the MSB
- 2's complement:  $v = -8y_3 + 4y_2 + 2y_1 + y_0$
- Unsigned X3:0  $u = +8x_3 + 4x_2 + 2x_1 + x_0$
- Offset Binary:  $v = +8x_3 + 4x_2 + 2x_1 + x_0 - 8 = u - 8$

## Signed number DAC



- Collect up all the unused currents from the R-2R ladder:
  - Total current into the ladder =  $16I_0$
  - Hence total current out of the ladder =  $16I_0$
  - Hence unused currents add up to  $(16 - X3:0)I_0$
- Send unused currents into a current mirror to reverse direction
- Add to original current to give  $2(X3:0-8)I_0$ .
- If  $Y3:0$  is a signed 2's complement number,  $v$ , we set  $\{X3, X2, X1, X0\}$  to  $\{!Y3, Y2, Y1, Y0\}$  which gives  $v = u - 8$  where  $u$  is  $X3:0$  as an unsigned number.
- Output current is now  $2yI_0$
- To invert  $Y3$ , we can just reverse the switch contacts.

## Current Mirror



The lower op-amp acts as a current mirror:

- Input current  $B$  all flows through the feedback resistor.
- Hence  $V_x = -BR$  since  $-ve$  input is a virtual earth.
- Hence second resistor has a voltage of  $BR$  across it since  $-ve$  input of 2nd op-amp is also a virtual earth.
- Hence current through second resistor is  $B$

Thus  $V_{OUT} = -(A - B) R_F$

Alternatively, in an integrated circuit, use a long-tailed pair or Wilson current mirror.

## Quiz

- Why is a weighted-resistor DAC impractical for a 16-bit converter?
- What is a *multiplying* DAC ?
- Why is a current mirror circuit so-called?
- What is the value of the bit pattern 1001 in the following notations: (a) unsigned binary, (b) two's complement binary, (c) offset binary ?
- How do you convert a number from offset binary to two's complement notation ?

## Lecture 11

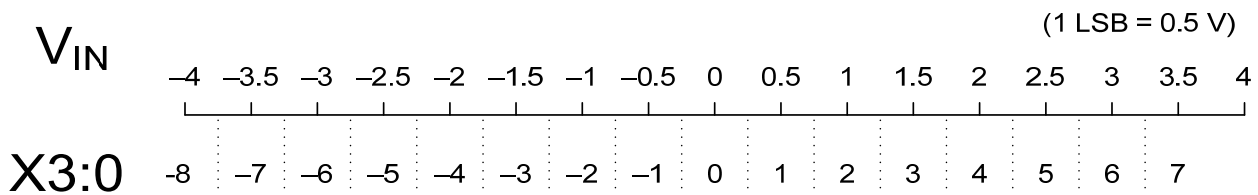
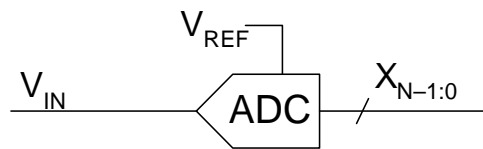
# Analog-to-Digital Conversion (1)

### Objectives

- Understand the relationship between the continuous input signal to an Analog-to-Digital converter and its discrete output
- Understand the source and magnitude of quantisation noise
- Understand how a flash converter works
- Understand how the use of dither can improve resolution and decorrelate the quantization noise



## Analog to Digital Conversion



Converters with  $\pm$ ve input voltages are called *bipolar* converters and usually round ( $V_{IN} \div 1\text{LSB}$ ) to the *nearest* integer.

$$X = \text{round}\left(\frac{V_{IN}}{1 \text{ LSB}}\right)$$

Example:

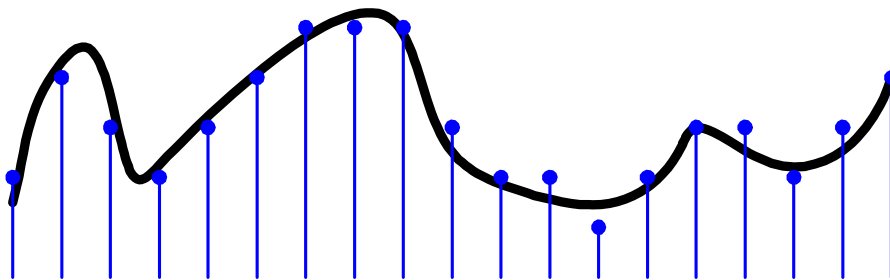
If 1 LSB = 0.5 V, then  $V_{IN} = 2.8 \text{ V}$  will be converted to:

$$X = \text{round}\left(\frac{2.8}{0.5}\right) = \text{round}(5.6) = 6$$

Analog to digital conversion destroys information: we convert a range of input voltages to a single digital value.

## Sampling

To process a continuous signal in a computer or other digital system, you must first sample it:



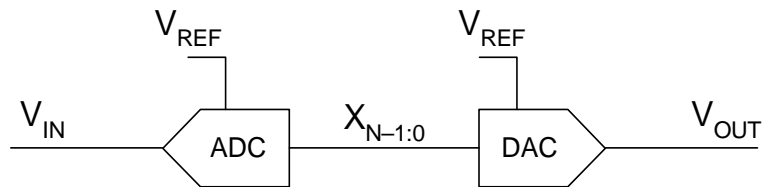
### Time Quantisation

- Samples taken (almost always) at regular intervals: sample frequency of  $f_{\text{samp}}$ .
- This causes *aliasing*: A frequency of  $f$  is indistinguishable from frequencies  $k f_{\text{samp}} \pm f$  for all integers  $k$ .
- No information lost if signal contains only frequencies below  $\frac{1}{2}f_{\text{samp}}$ . This is the *Nyquist limit*.

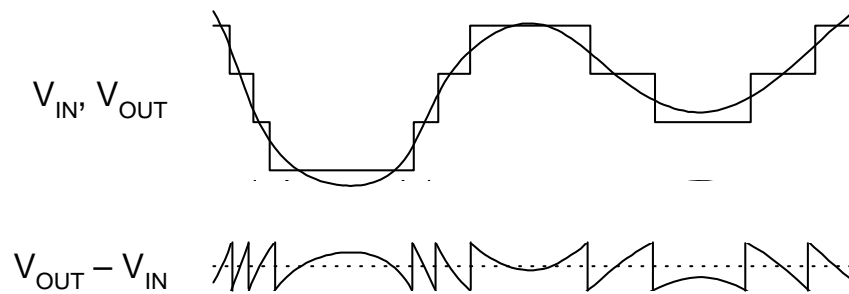
### Amplitude Quantisation

- Amplitude of each sample can only take one of a finite number of different values.
- This adds quantisation noise: an irreversible corruption of the signal.
- For low amplitude signals it also adds distortion. This can be eliminated by adding dither before sampling.

## Quantisation Noise



$V_{OUT}$  is restricted to discrete levels so cannot follow  $V_{IN}$  exactly. The error,  $V_{OUT} - V_{IN}$  is the quantisation noise and has an amplitude of  $\pm \frac{1}{2}$  LSB.



If all error values are equally likely, the RMS value of the quantisation noise is

$$\sqrt{\int_{-1/2}^{+1/2} x^2 dx} = \frac{1}{\sqrt{12}} = 0.3 \text{ LSB}$$

### Signal-to-Noise Ratio (SNR) for an n-bit converter

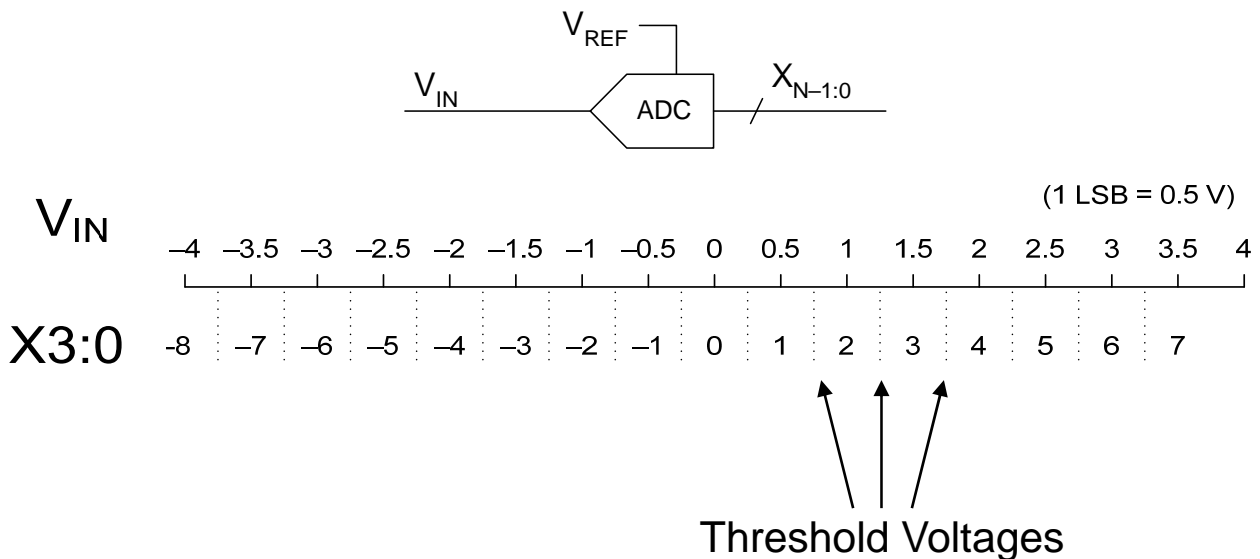
Ratio of the maximum sine wave level to the noise level:

- Maximum sine wave has an amplitude of  $\pm 2^{n-1}$  which equals an RMS value of  $0.71 \times 2^{n-1} = 0.35 \times 2^n$ .

- SNR is:

$$20 \log_{10} \left( \frac{0.35 \times 2^n}{0.3} \right) = 20 \log_{10} (1.2 \times 2^n) = 1.8 + 6n \text{ dB}$$

## Threshold Voltages



Each value of  $X$  corresponds to a range of values of  $V_{IN}$ .

The voltage at which  $V_{IN}$  switches from one value of  $X$  to the next is called a *threshold voltage*.

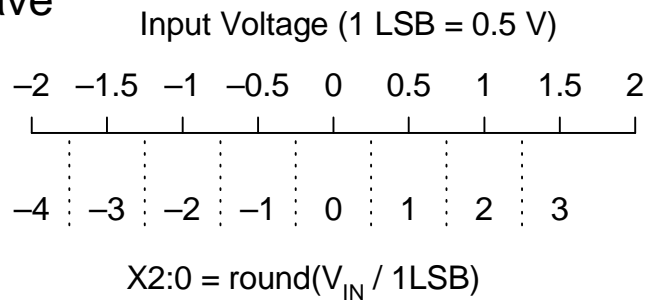
The task of an A/D converter is to discover which of the voltage ranges  $V_{IN}$  belongs to. To do this, the converter must compare  $V_{IN}$  with the threshold voltages.

The threshold voltages corresponding to  $X$  are at  $(X \pm \frac{1}{2})$  LSB

## Flash A/D Converter

For an  $n$ -bit converter we have

$2^n - 1$  *threshold voltages*.

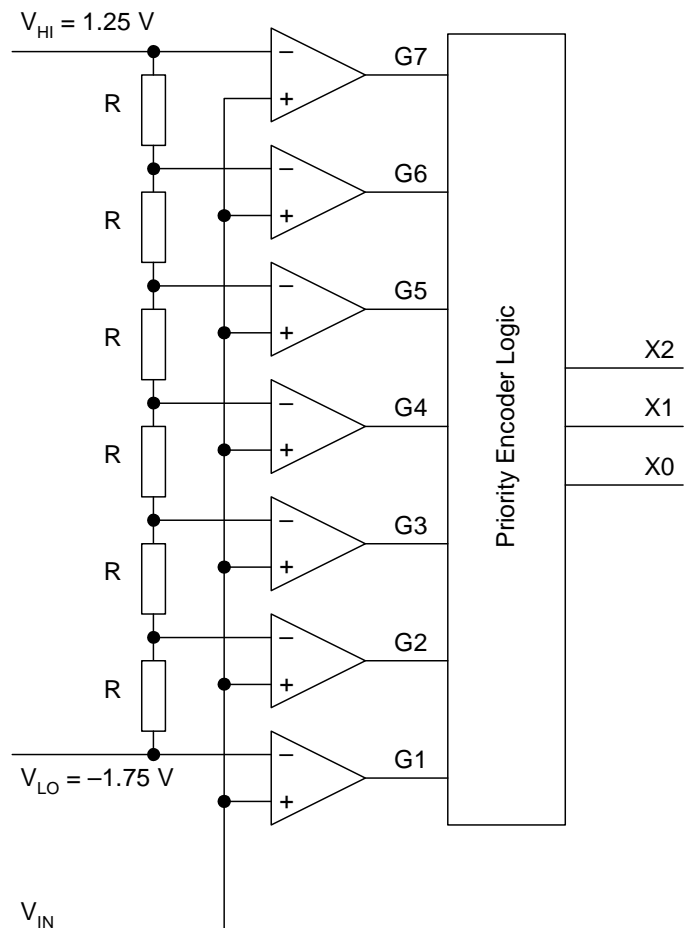


Use  $2^n - 1$  comparators:

Resistor chain used to generate threshold voltages.


Priority encoder logic must determine the highest  $G_n$  input that equals 1.

12-bit converter needs 4095 comparators on a single chip!



## Priority Encoder

G7:1 can have  $2^7$  possible values but only 8 will occur:

	<b>G7:1</b>	<b>X2:0</b>	<b>Example: G2 • !G4</b>
$V_{IN} > 1.25:$	1111111	011 =+3	0
	0111111	010 =+2	0
	0011111	001 =+1	0
	0001111	000 =+0	0
	0000111	111 =-1	1
	0000011	110 =-2	1
	0000001	101 =-3	0
$V_{IN} < -1.75:$	0000000	100 =-4	0
	 G4 G2		

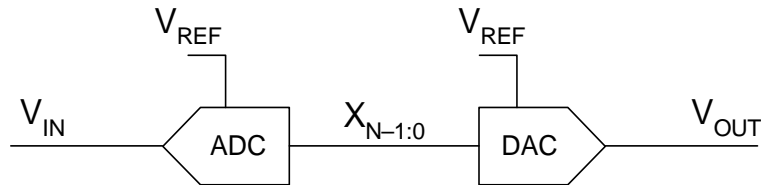
By inverting one comparator output and ANDing it with another one, we can generate a signal that is high for any group of consecutive X values.

- Example:  $G2 \bullet !G4$  is high for  $-2 \leq X \leq -1$

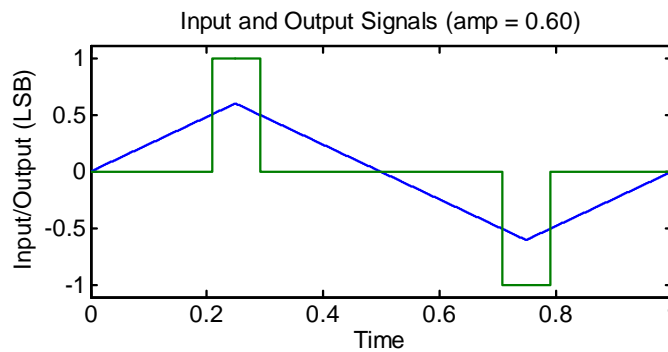
Hence we can generate each of X2, X1 and X0 by ORing together a number of such terms:

- $X2 = !G4$
- $X1 = G6 + G2 \bullet !G4$
- $X0 = G7 + G5 \bullet !G6 + G3 \bullet !G4 + G1 \bullet !G2$

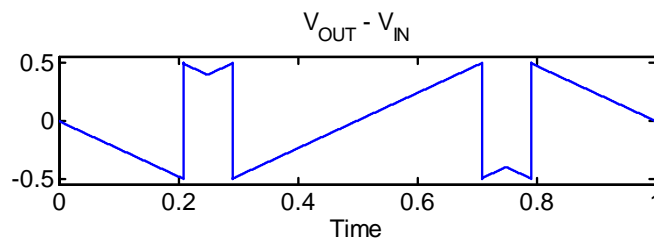
## Quantisation Distortion for Small Signals



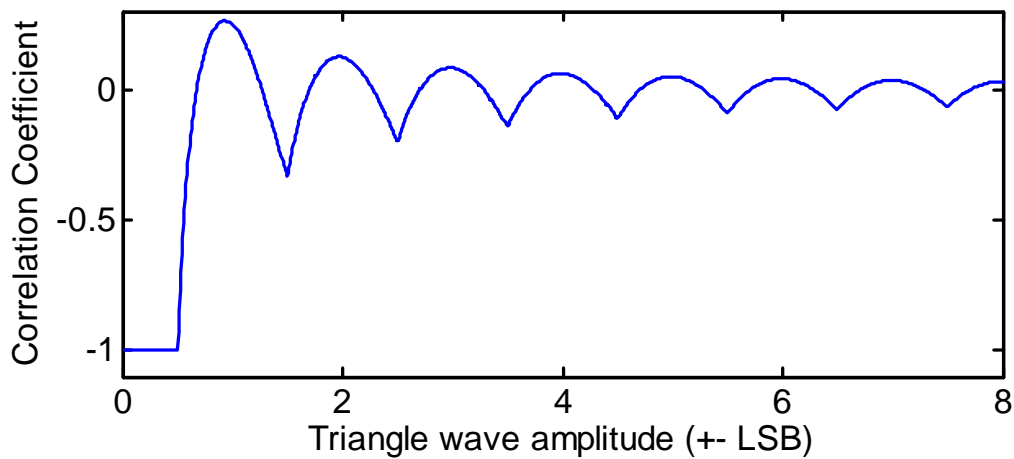
If  $V_{IN}$  is a low amplitude triangle wave ( $\pm 0.6$  LSB):



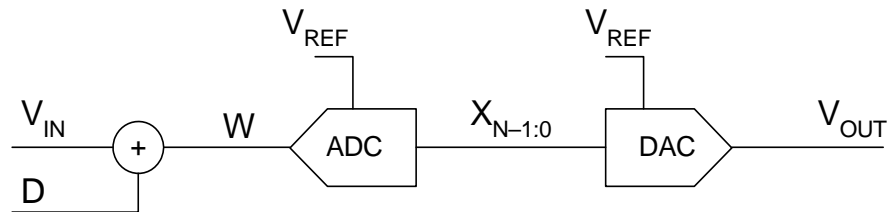
Error has strong negative correlation with  $V_{IN} \Rightarrow$  distortion



Correlation coefficient  $\rightarrow 0$  at high amplitudes:

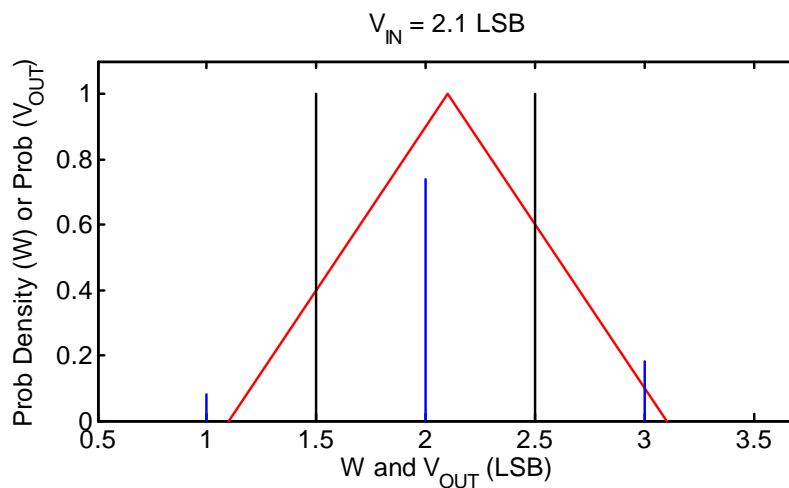


## Dither



Dither,  $D$ , is a random noise with a triangular probability density.

If  $V_{IN} = 2.1$  LSB, then  $W$  has a triangular distribution and  $V_{OUT}$  takes three possible values:



$$V_{OUT} = \begin{cases} 1 & p(1) = p(W < 1.5) = 0.08 \\ 2 & p(2) = p(1.5 < W < 2.5) = 0.74 \\ 3 & p(3) = p(W > 2.5) = 0.18 \end{cases}$$

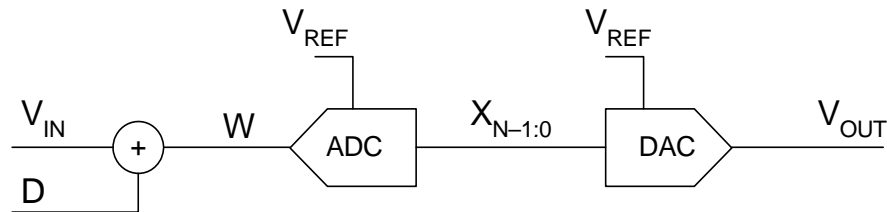
$$E(V_{OUT}) = 1 \times 0.08 + 2 \times 0.74 + 3 \times 0.18 = 2.1$$

$$Var(V_{OUT}) = 1^2 \times 0.08 + 2^2 \times 0.74 + 3^2 \times 0.18 - 2.1^2 = 0.25$$

$E(V_{OUT}) = V_{IN}$  and  $Var(V_{OUT}) = 0.25$  for all values of  $V_{IN}$



## Effects of Dither



### Dither should be added to a signal

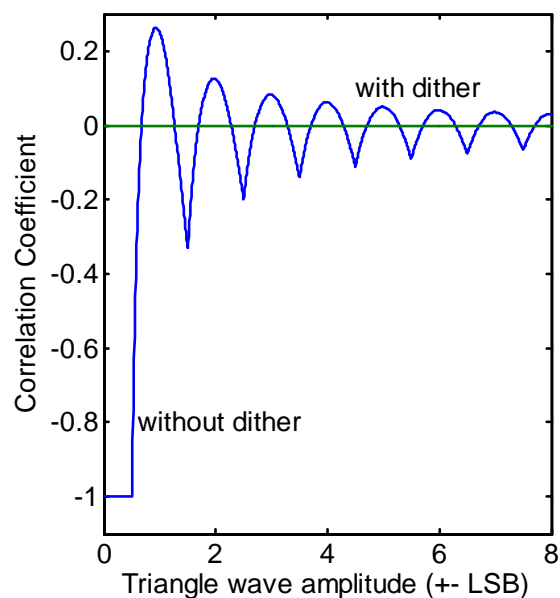
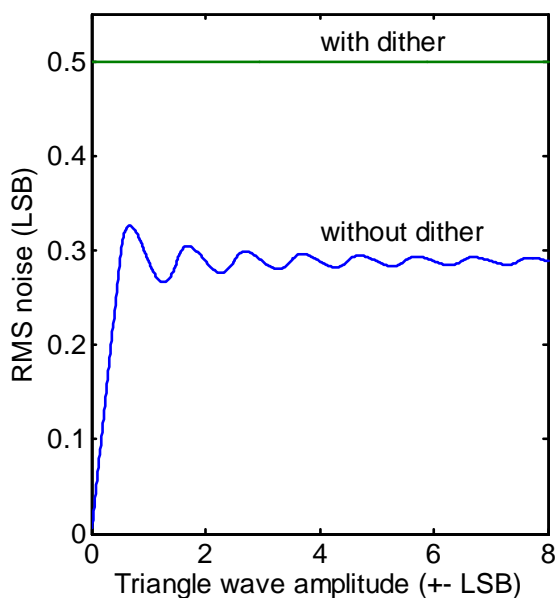
- before an ADC
- before reducing digital precision (e.g. 16 to 8 bits)
- Triangular pdf of amplitude  $\pm 1$  LSB at new precision

### Good consequences

- Quantisation noise level is constant independent of  $V_{IN}$
- Quant noise is uncorrelated with  $V_{IN} \Rightarrow$  no distortion
- Signal variations are preserved even when  $< 1$  LSB

### Bad consequence

- RMS quantisation noise increases from 0.3 to 0.5 LSB



## Quiz

- What is a *bipolar* A/D converter ?
- What is the amplitude of the quantisation noise introduced by an A/D converter ?
- How many threshold voltages are there in an  $n$ -bit converter ?
- What is the function of a priority encoder ?
- What is the level of quantisation noise for large signal variations ?
- What are the good and bad consequences of adding dither to a signal before conversion to digital ?

## Lecture 12

# **Analog-to-Digital Conversion (2)**

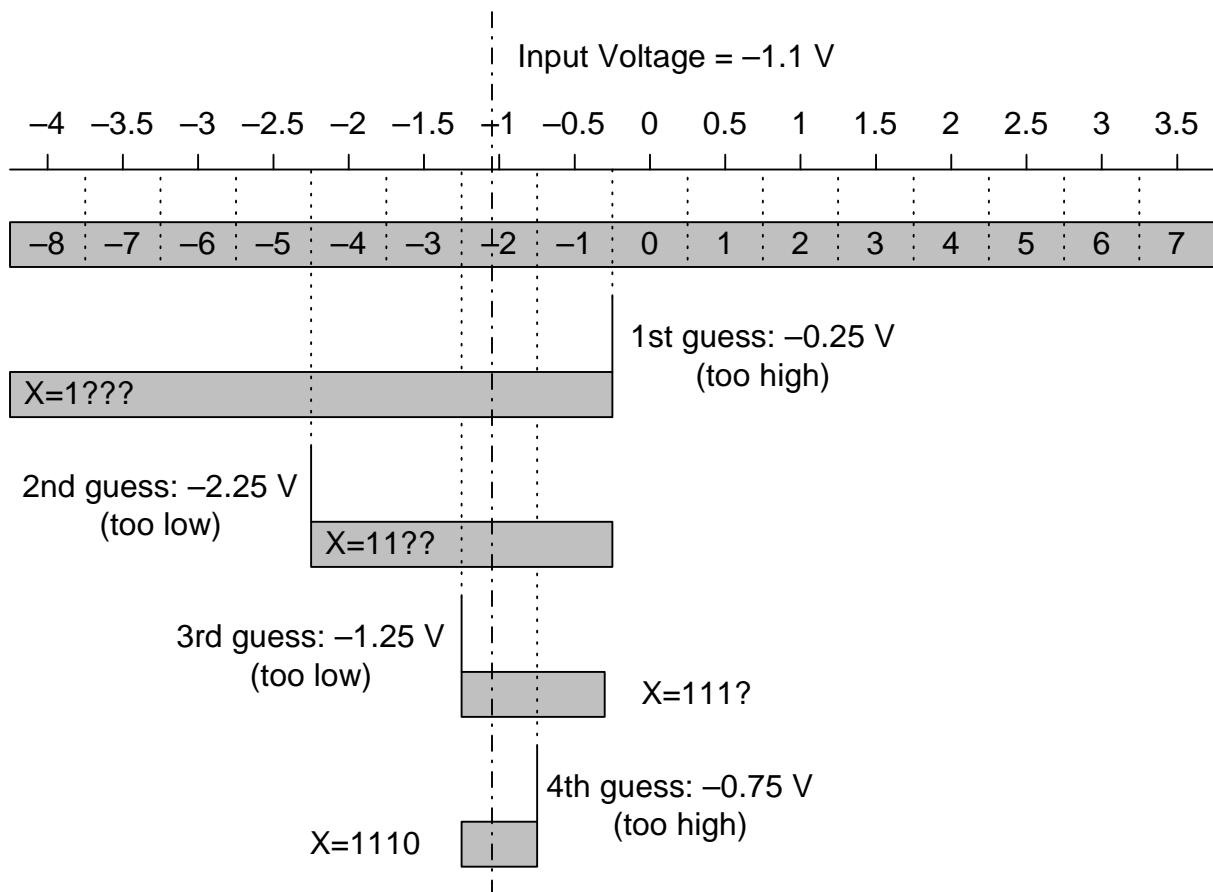
### Objectives

- Understand the principles behind a successive approximation converter
- Understand how a successive approximation converter can be implemented using a state machine
- Understand the need for using a sample/hold circuit with a successive approximation converter
- Understand the origin of glitches at the output of a DAC and how they can be avoided.

## Successive Approximation Converter

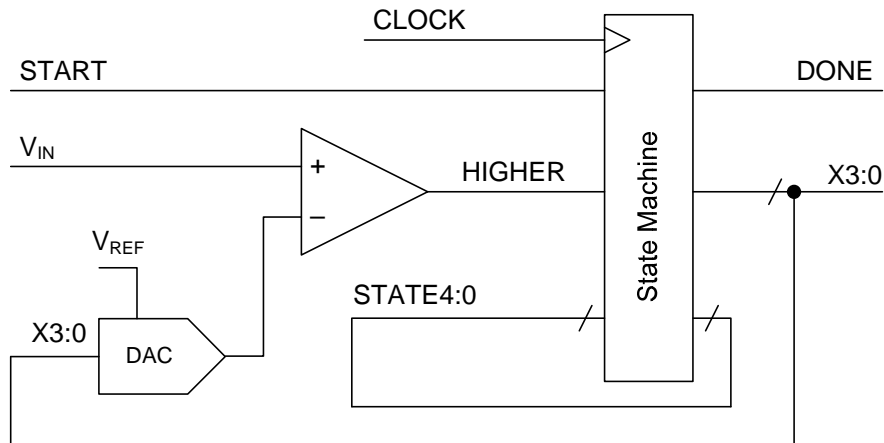
Make successive guesses and use a comparator to tell whether your guess is too high or too low.

Each guess determines one bit of the answer and cuts the number of remaining possibilities in half:



Use a DAC to generate the threshold voltages and a state machine to create the sequence of guesses. A DAC input of  $n$  generates the threshold between  $n-1$  and  $n$  which equals  $(n-\frac{1}{2}) \times 1\text{ LSB}$

## Successive Approximation ADC

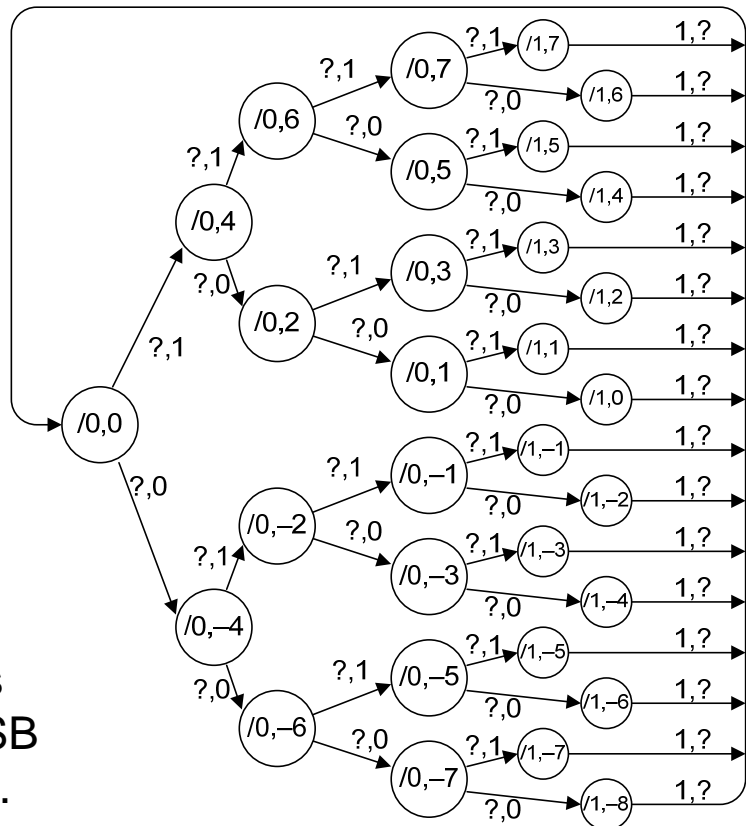


### State Diagram:

A DAC input of  $n$  must generate the *threshold* between  $n-1$  and  $n$ .

When the final column of states is reached, DONE goes high and the answer is X3:0.

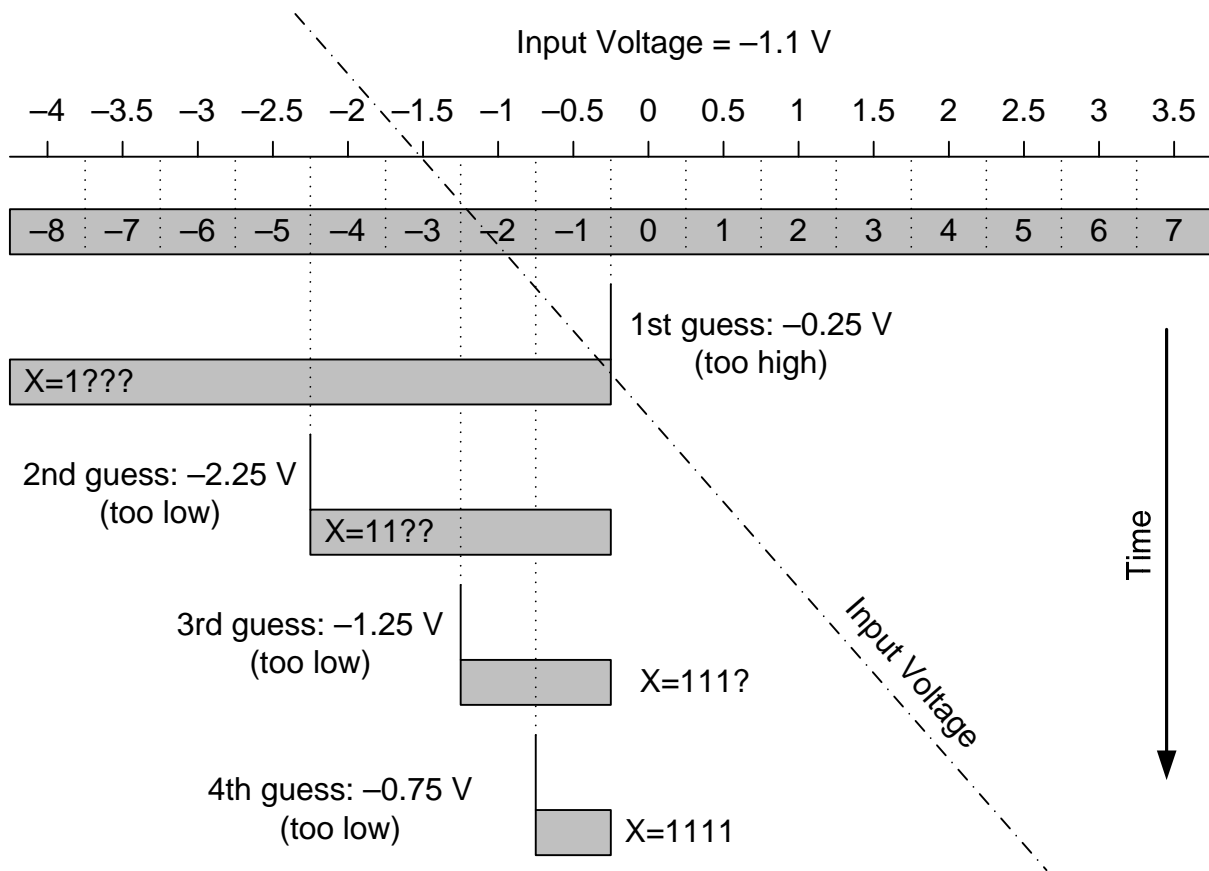
Note that it is possible to number the 31 states so that DONE is the MSB and X3:0 are the 4 LSB.



I/O Signals: START,HIGHER/DONE,X3:0

## Need for Sample/Hold

If the input voltage changes during conversion, the result is biased towards its initial value because the most significant bits are determined first.

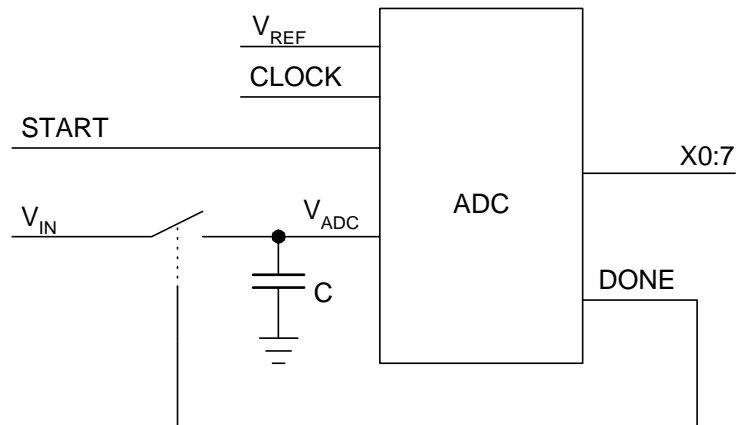


Increasing voltages will tend to be converted to values ending in ...111. Decreasing voltages will tend to be converted to values ending in ...000.

Consequences:

reduced precision, uncertain sample instant.

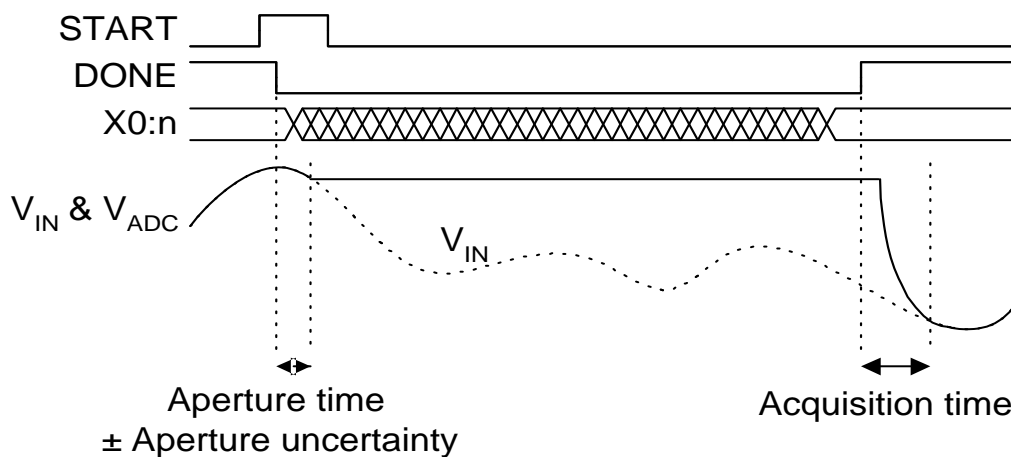
## A/D conversion with sample/hold



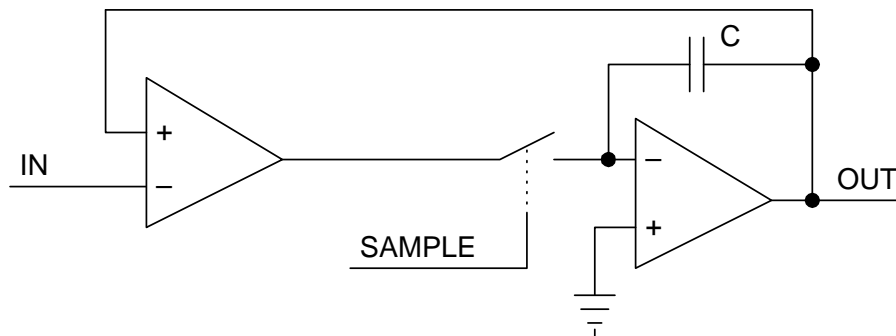
Input switch is opened during the conversion so  $V_{ADC}$  remains constant.

Choice of  $C$  is a compromise:

- Big  $C$  keeps constant voltage despite leakage currents since  $dV/dt = I_{leakage}/C$
- Small  $C$  allows faster acquisition time for any given input current since  $dV/dt = I_{in}/C$ .



## Sample/Hold Circuit



When switch is open:

- Leakage currents through open switch and op-amp input will cause output voltage to drift up or down.
- Choose capacitor large enough that this drift amounts to less than 0.5 LSB during the time for a conversion
- Converters with high resolution or long conversion times need larger capacitors

When switch closes:

- Charge rate of capacitor is limited by the maximum op-amp output current. This determines the *acquisition time*: to acquire the signal to within  $\frac{1}{2}$ LSB. It is typically of the same order as the conversion time.

Value of C is a compromise: big C gives slow acquisition, small C gives too much drift.



## Other types of Converter

### Sampling ADC

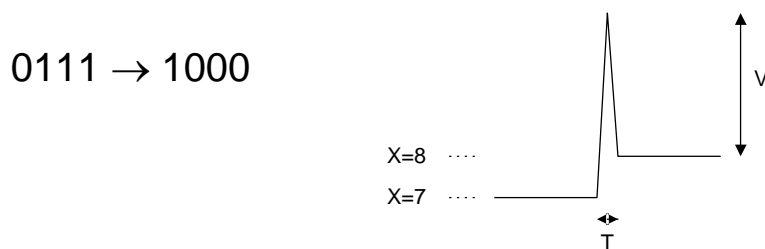
Many A/D converters include a sample/hold within them: these are *sampling* A/D converters.

### Oversampling DAC and ADC

*Oversampling* converters (also known as  $\Sigma\Delta$  or  $\Delta\Sigma$  converters) sample the input signal many times for each output sample. By combining digital averaging with an error feedback circuit they can obtain up to 20 bits of precision without requiring a high accuracy resistor network (hence cheaper). A typical oversampling ratio is 128x, i.e. the input is sampled at 6.4MHz to give output samples at 50 kHz. Most CD players use an oversampling DAC.

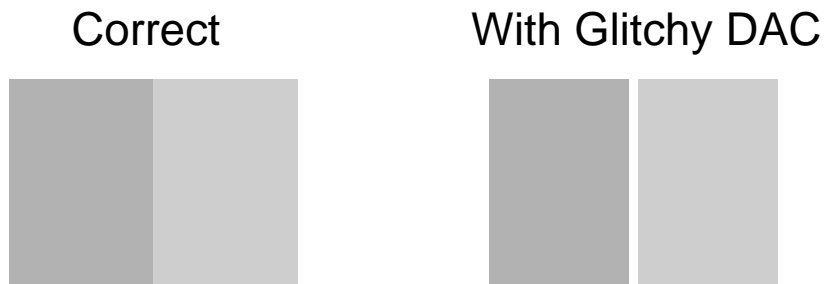
## Glitches in DAC output voltages

Switches in DAC operate at different speeds  $\Rightarrow$  output glitches occur when several input bits change together:



Cannot remove glitches: low pass filtering merely spreads out the glitch: the *glitch energy* –  $V \times T$  remains constant.

Glitches are very noticeable on a video display:

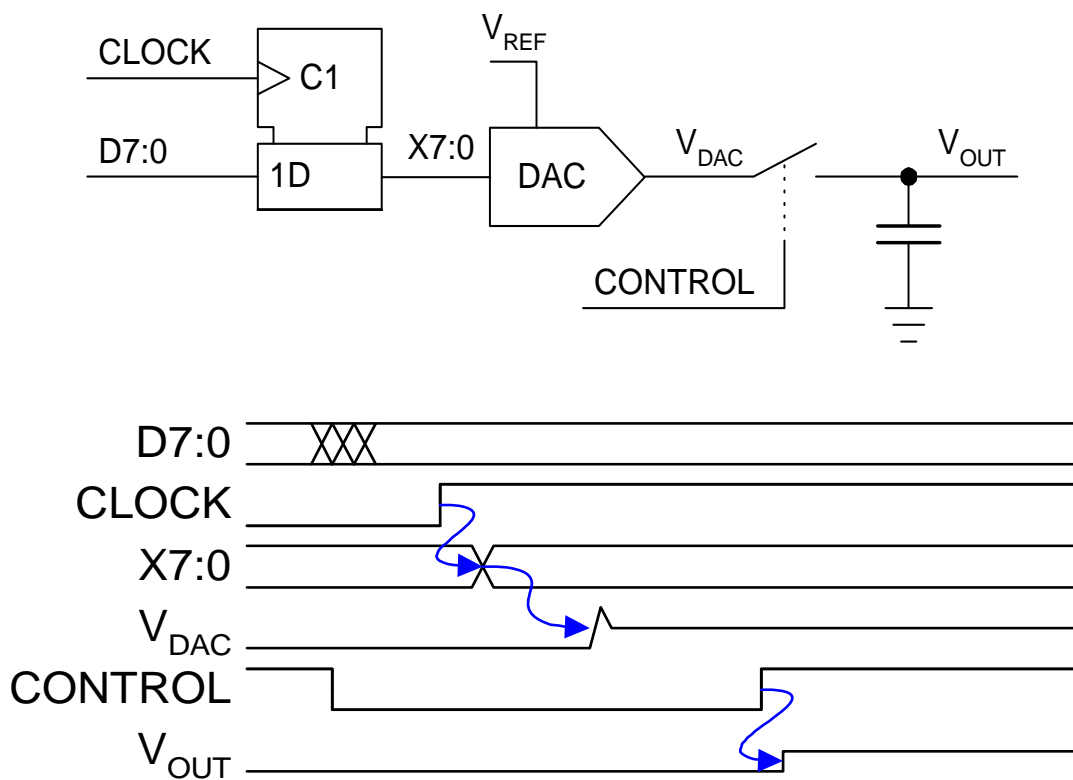


**Solution:** We use a sample/hold circuit to isolate the output from the DAC while the glitch is happening.

## Deglitching

To minimize the effect of glitches:

- Use a register to make inputs change as simultaneously as possible
- Use a sample/hold circuit to disconnect the DAC output while it is changing



## Summary

### D/A Converters:

- Weighted resistor: very fast (no op-amp), each bit can have an arbitrary weight, not good for big numbers.
- R-2R ladder: used for most converters, switch currents rather than voltages for higher speed. Multiplying DAC has an analog input as well.
- $\Sigma\Delta$  converters used for audio: very good linearity.

### A/D Converters:

- Flash converter: very fast (down to 1 ns), low precision (8 bits max), expensive and power hungry. A “pipeline” converter uses a DAC to subtract the converted value and measures the difference with another flash converter .
- Successive Approximation: medium speed (down to 0.1  $\mu$ s), need to use sample/hold circuit to avoid input changing during conversion.
- $\Sigma\Delta$  converters dominate the medium to low speed market (down to 0.5  $\mu$ s). Long been standard for audio: very good linearity (up to 24 bits). Very high speed sampling at low precision with dither, followed by low-pass digital filter and sub-sampling to desired sample rate.

## Quiz

- How many voltage comparisons are made by an  $n$ -bit successive approximation converter during the course of a conversion ?
- What is a *multiplying* DAC ?
- Why does the DAC in an  $n$ -bit successive approximation converter only need to generate  $2^n - 1$  different values rather than  $2^n$  ?
- If a 12-bit successive approximation converter is used without a sample/hold, which of the output values 127, 128 and 129 are likely to occur least frequently ?
- What is the aperture uncertainty of a sample/hold circuit ?
- What two effects determine the acquisition time of a sample/hold circuit ?
- What happens if you try to improve a glitchy signal using a low-pass filter ?

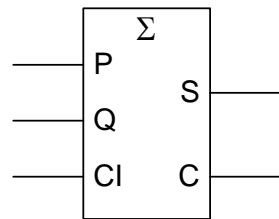
## Lecture 13

# Adder Circuits

### Objectives

- Understand how to add both signed and unsigned numbers
- Appreciate how the delay of an adder circuit depends on the data values that are being added together

## Full Adder



$$\begin{array}{r} P \\ Q \\ + CI \\ \hline C \quad S \end{array}$$

Output is a 2-bit number counting how many inputs are high

P	Q	CI	C	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

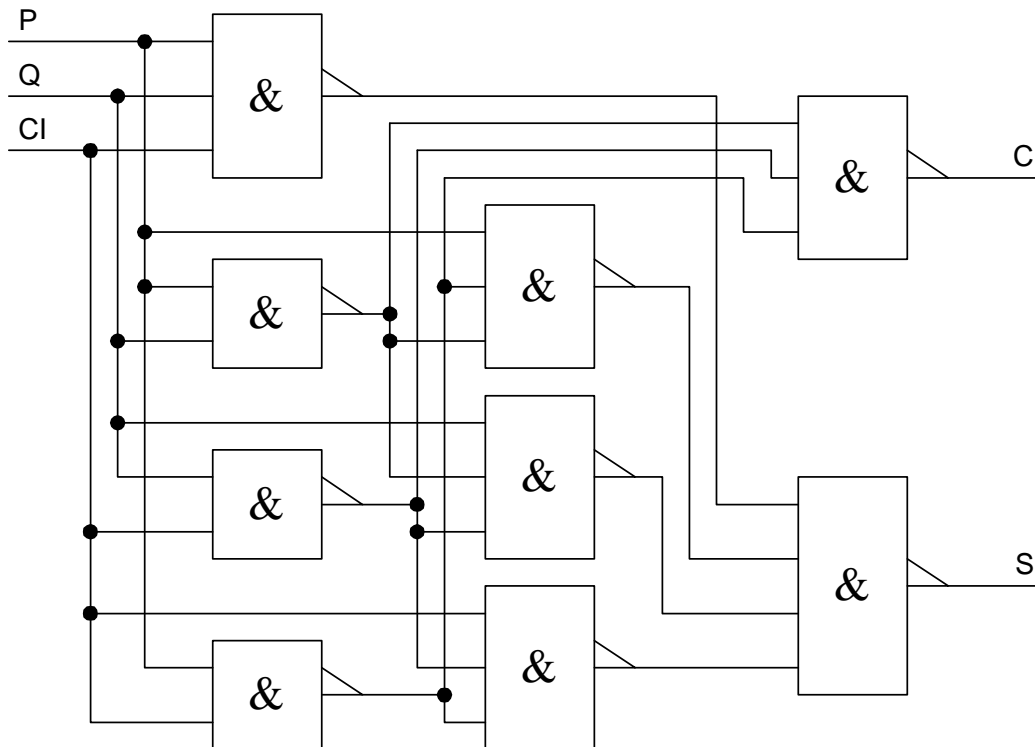
$$C = P \cdot Q + P \cdot CI + Q \cdot CI$$

$$S = P \oplus Q \oplus CI$$

- Symmetric function of the inputs
- Self-dual: Invert all inputs  $\Rightarrow$  invert all outputs  
If  $k$  inputs high initially then  $3-k$  high when inverted  
Inverting all bits of an  $n$ -bit number make  $x \rightarrow 2^n - 1 - x$
- Note:  $P \oplus Q \oplus CI = (P \oplus Q) \oplus CI = P \oplus (Q \oplus CI)$

## Full Adder Circuit

9-gate full-adder NAND implementation (do not memorize)



Propagation delays:

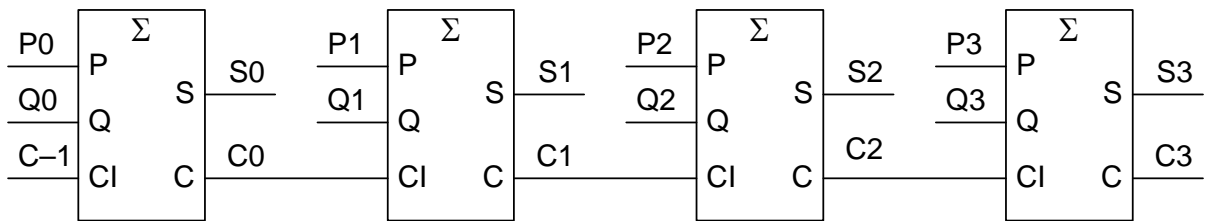
From	To	Delay
P,Q or Cl	S	3
P,Q or Cl	C	2

Complexity: 25 gate inputs  $\Rightarrow$  50 transistors but use of complex gates can reduce this somewhat.



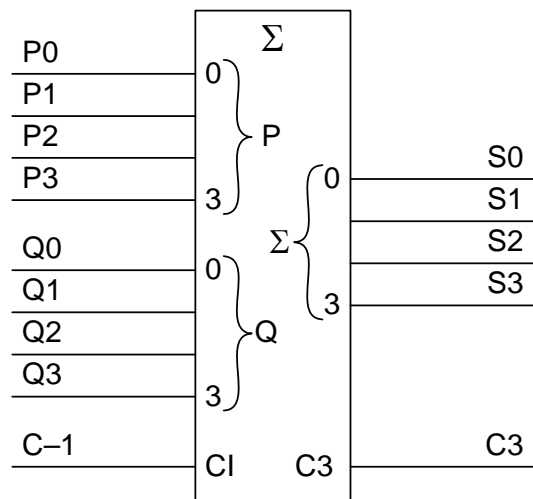
## N-bit adder

We can make an adder of arbitrary size by cascading full adder sections:



The main reason for using 2's complement notation for signed numbers is that:

**Signed and unsigned** numbers can use identical circuitry



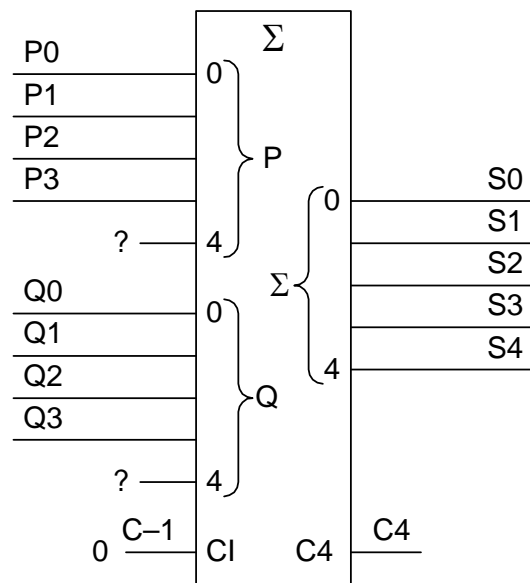
## Adder Size Selection

The number of bits needed in an adder is determined by the range of values that can be taken by its **output**.

If we add two 4-bit numbers, the answer can be in the range:

- 0 to 30 for *unsigned* numbers
- -16 to +14 for *signed* numbers

In both cases we need a 5-bit adder to avoid any possibility of overflow:



We need to expand the input numbers to 5 bits. How do we do this ?

## Expanding Binary Numbers

### Unsigned numbers

Expand an unsigned number by adding the appropriate number of 0's at the MSB end:

5	0101	00000101
13	1101	00001101

### Signed numbers

Expand a signed number by duplicating the MSB the appropriate number of times:

5	0101	00000101
-3	1101	11111101

This is known as *sign extension*

---

## Shrinking Binary Numbers

### Unsigned

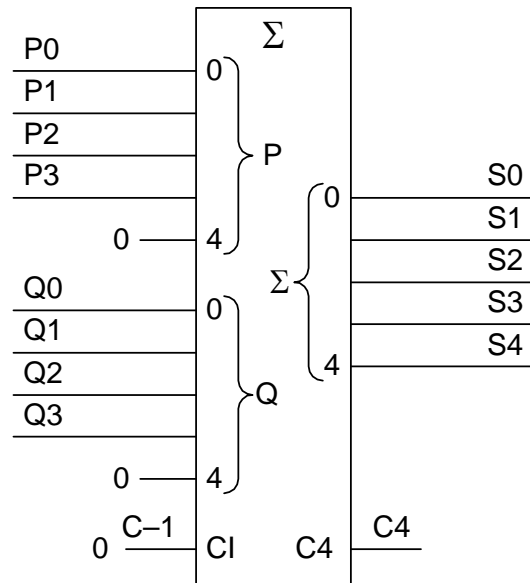
Can delete any number of bits from the MSB end so long as they are all 0's.

### Signed

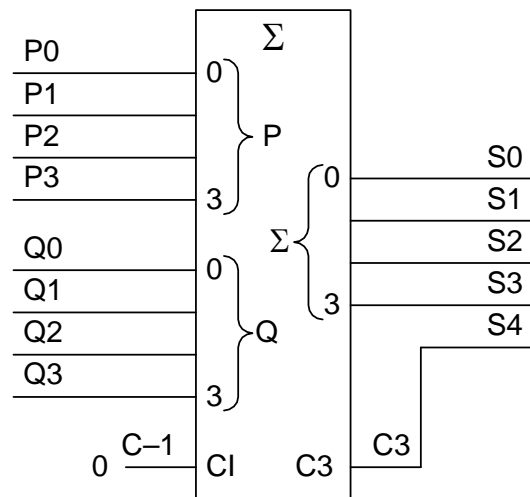
Can delete any number of bits from the MSB end so long as they are all the same as the MSB that remains.

## Adding Unsigned Numbers

To avoid overflow, we use a 5-bit adder:



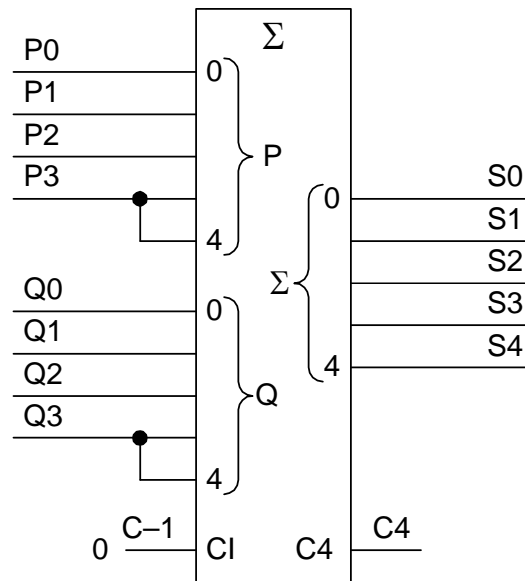
The MSB stage is performing the addition:  $0 + 0 + C_3$ . Thus  $S_4$  always equals  $C_3$  and  $C_4$  always equals 0.



We can use a 4-bit adder with  $C_3$  as an answer bit.

## Adding Signed Numbers

To avoid overflow, we use a 5-bit adder:



This is different from the unsigned case because P4 and Q4 are no longer constants. [We cannot simplify this circuit by removing the MSB stage.](#)

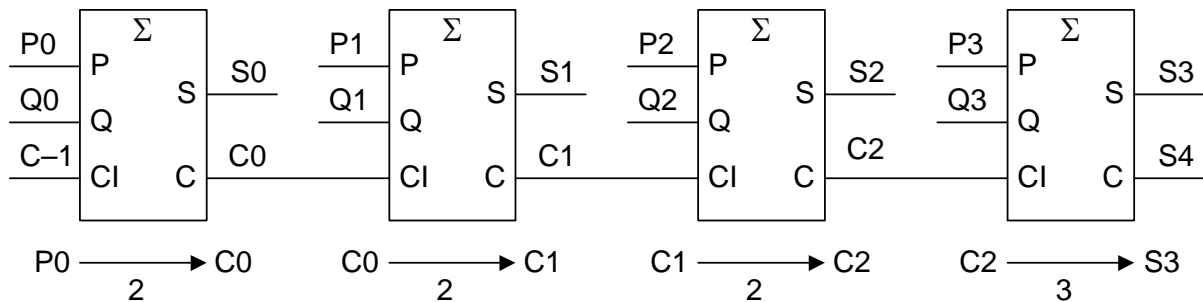
If P and Q have different signs then S4 will not equal C3.

e.g. P=0000, Q=1111  
 Unsigned P+Q=01111, Signed P+Q=11111

Some minor simplifications are possible:

- If the C4 output is not required, the circuitry that generates it can be removed.
- S4 can be generated directly from P3, Q3 and C3 which reduces the circuitry needed for the last stage.

## Adder Propagation Delay



Delays within each stage (in gate delays):

$$P, Q, CI \rightarrow S = 3 \quad P, Q, CI \rightarrow C = 2$$

Worst-case delay is:

$$P_0 \rightarrow C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow S_3 = 3 \times 2 + 3 = 9$$

Note: We also have  $Q_0 \rightarrow S_3 = 9$  and  $C_{-1} \rightarrow S_3 = 9$

For an N-bit adder, the worst delay is  $(N-1) \times 2 + 3 = 2N+1$

Example of worst case delay:

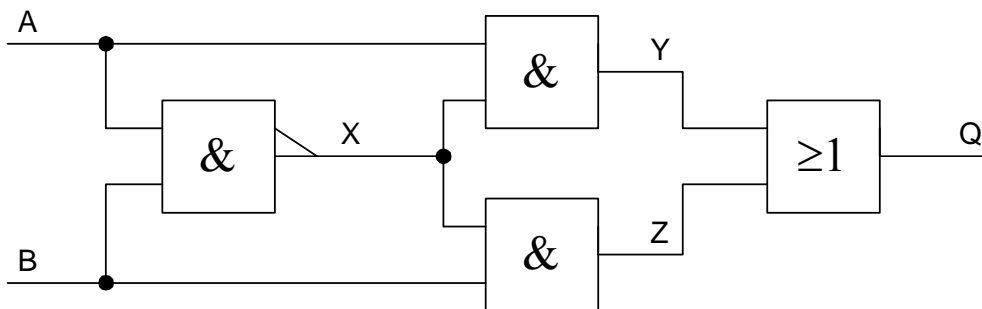
- Initially:  $P_{3:0}=0000, Q_{3:0}=1111 \Rightarrow S_{4:0}=01111$
- Change to:  $P_{3:0}=0001, Q_{3:0}=1111 \Rightarrow S_{4:0}=10000$

## Delays are Data-Dependent

To determine the delay of a circuit, we need to specify:

1. The circuit
2. The initial value of all the inputs
3. Which of the inputs changes

Example: What is the propagation delay  $A \rightarrow Q$  ?



Answer 1 ( $B=0$ ):

- Initially:  $A=0, B=0 \Rightarrow X=1, Y=0, Z=0, Q=0$
- Then:  $A \uparrow \Rightarrow Y \uparrow \Rightarrow Q \uparrow$                       2 gate delays

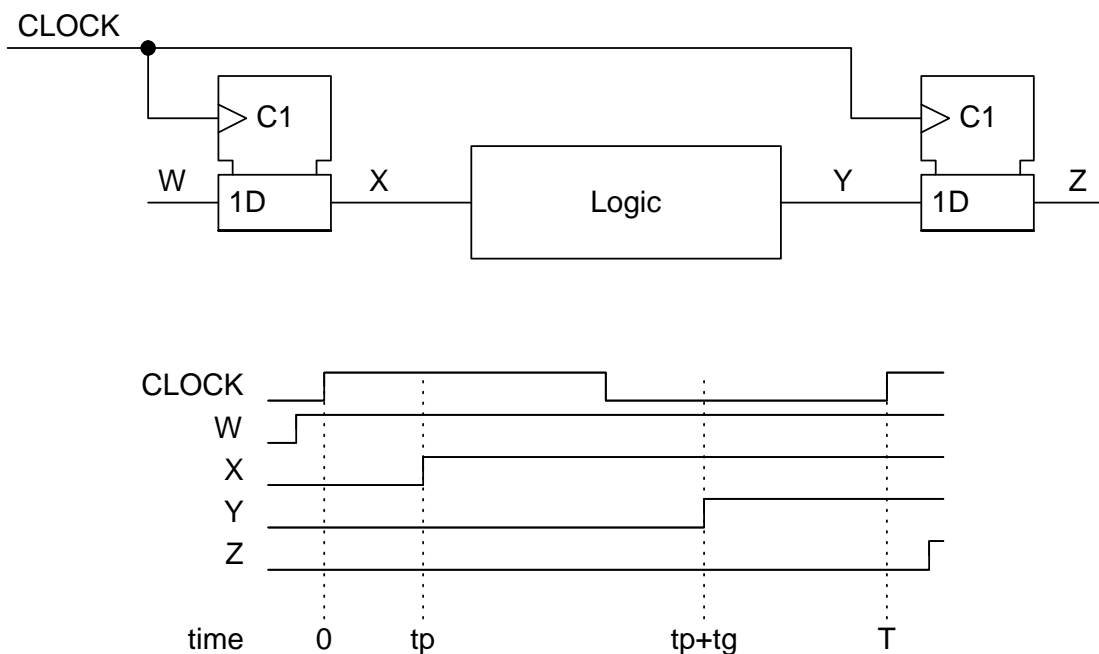
Answer 2 ( $B=1$ ):

- Initially:  $A=0, B=1 \Rightarrow X=1, Y=0, Z=1, Q=1$
- Then:  $A \uparrow \Rightarrow X \downarrow \Rightarrow Z \downarrow \Rightarrow Q \downarrow$                       3 gate delays

## Worst-Case Delays

We are normally interested only in the worst-case delay from a change in any input to any of the outputs.

The worst-case delay determines the maximum clock speed in a synchronous circuit:



$$tp + tg + ts < T$$

Since the clock speed must be chosen to ensure that the circuit always works, it is only the worst-case logic delay that matters.



## Quiz

1. In an full adder, why is it normally more important to reduce the delay from CI to C than to reduce the delay from P to S ?
2. How many bits are required to represent the number  $A+B$  if A and B are (a) 8-bit *unsigned* numbers or alternatively (b) 8-bit *signed* numbers.
3. How do you convert a 4-bit *signed* number into an 8-bit signed number ?
4. How do you convert a 4-bit *unsigned* number into an 8-bit signed number ?
5. How is it possible for the propagation delay of a circuit from an input to an output to depend on the value of the other inputs ?

## Lecture 14

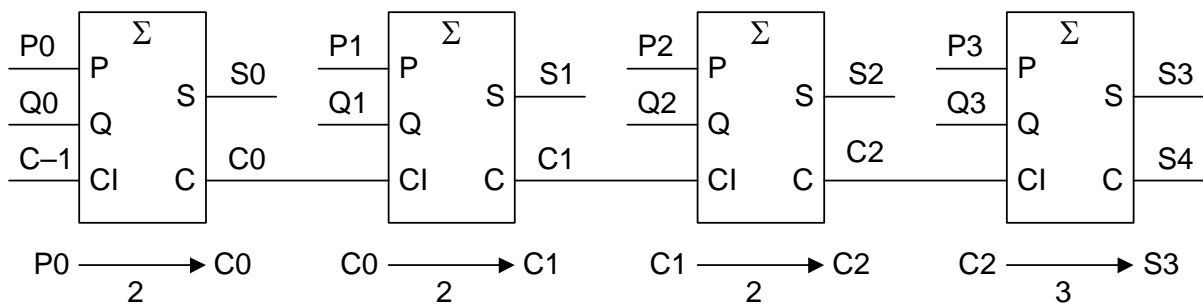
# Fast Adder Circuits (1)

### Objectives

- Understand how the propagation delay of an adder can be reduced by inverting alternate bits.
- Understand how the propagation delay of an adder can be reduced still further by means of carry lookahead.

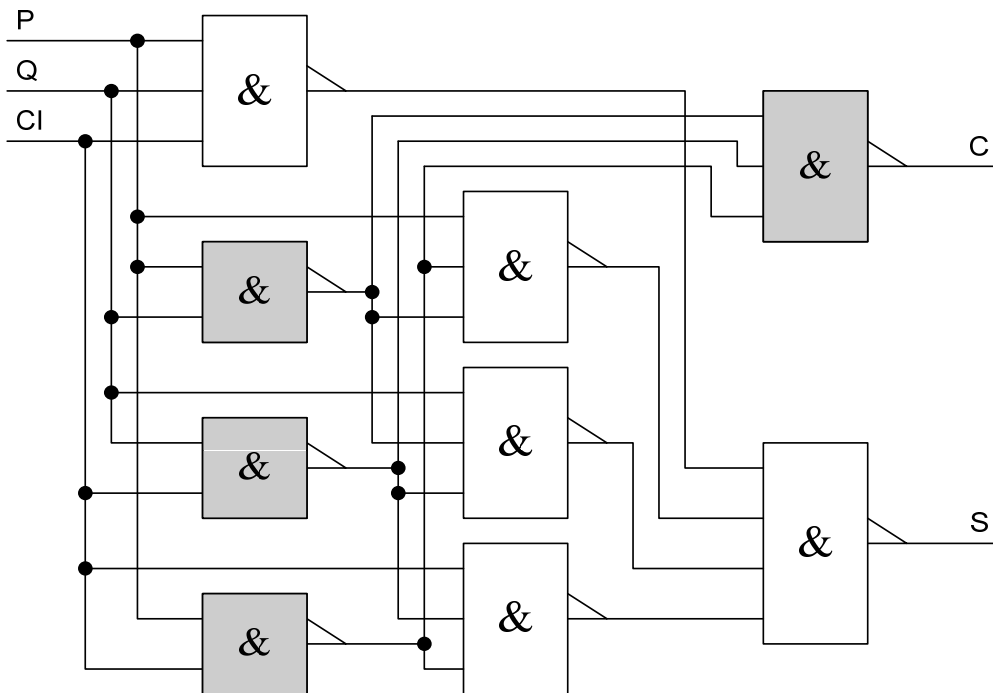
## Standard N-bit Adder

Delay of standard N-bit adder =  $2N+1$



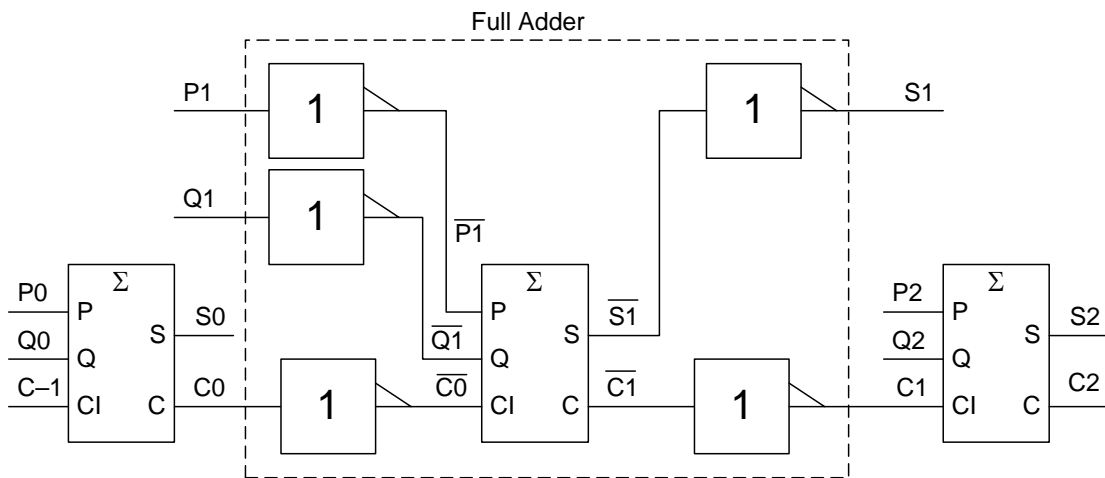
Delay of carry path within each full adder = 2

Carry path consists of three 2-input + one 3-input NANDs

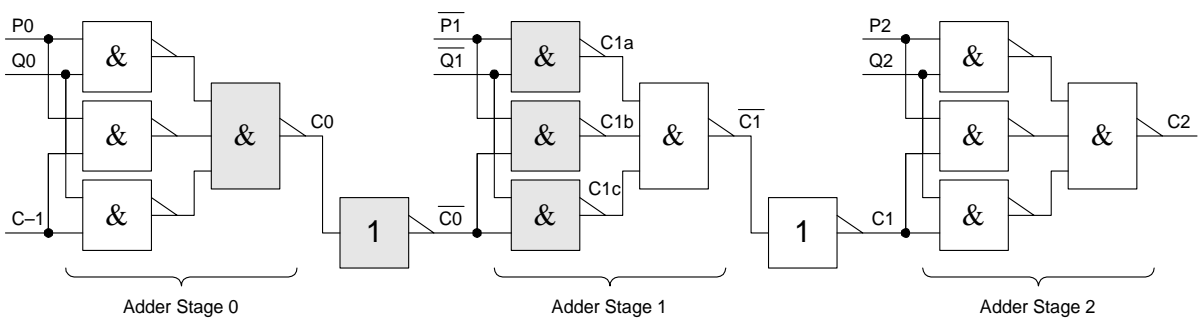


## Faster Adder Circuits: 1

Because a full-adder is *self-dual*, it will still work if for alternate stages we invert both the inputs and the outputs:

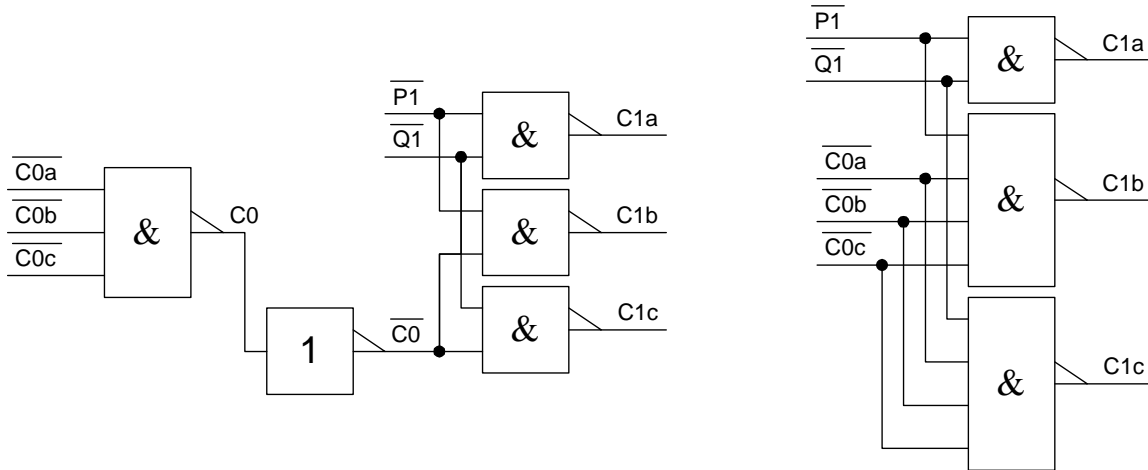


Now consider only the Carry signals:

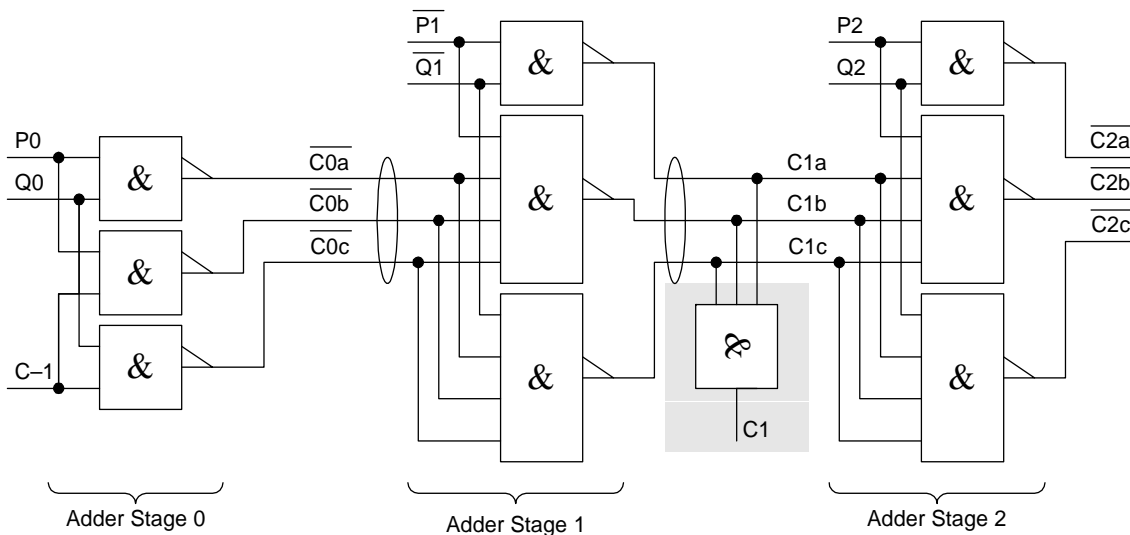


By merging the shaded gates we can reduce the delay to one gate per adder stage.

## Fast Adder Circuits: 1 (part 2)



We can merge the 3-NAND and inverter into the final column of gates as shown; this gives one delay per stage:



The signals  $C1a$ ,  $C1b$ ,  $C1c$  form an *AND-bundle*:  $C1$  is true only if all of them are high. We don't need the signal  $C1$  directly so the shaded gate can be omitted.

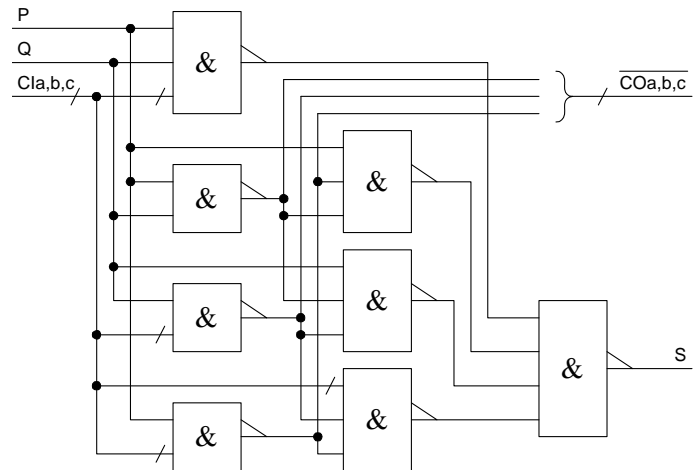
## Fast Adder Circuits: 1 (part 3)

### Even stages:

Delays:

$P, Q, CI \rightarrow S \quad 3$

$P, Q, CI \rightarrow C \quad 1$



30 gate inputs  $\Rightarrow$  60 transistors

### Odd stages:

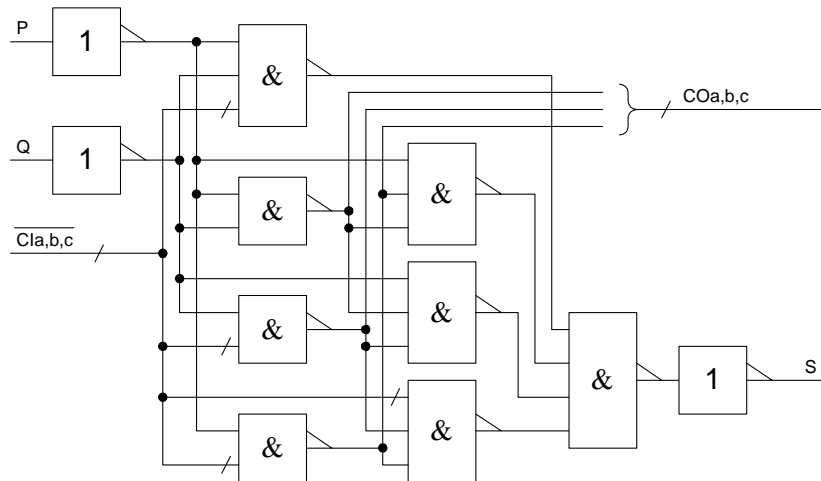
Delays:

$P, Q \rightarrow S \quad 5$

$P, Q \rightarrow C \quad 2$

$CI \rightarrow S \quad 4$

$CI \rightarrow C \quad 1$



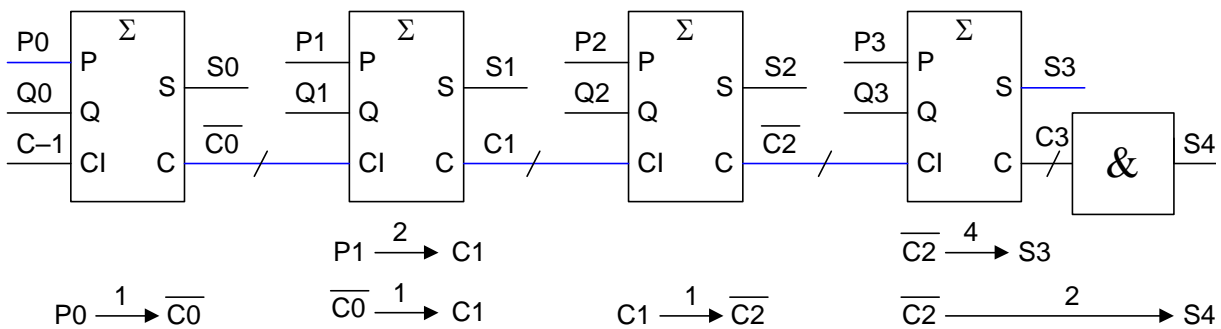
33 gate inputs  $\Rightarrow$  66 transistors

Bundles are denoted by a single wire with a / through it.

22% more transistors but twice as fast.

## Fast Adder Circuits: 1 (part 4)

For an N-bit adder we alternate the two modules (with a normalish first stage):



Worst case delay is:

$$P0 \rightarrow !C0 \rightarrow C1 \rightarrow !C2 \rightarrow S3 = 7 \text{ gate delays}$$

Note that:

- Delay to S4 is shorter than delay to S3
- Delay from P1 is the same as delay from P0
- Worst-case example:  
Initially: P3:0=0000, Q3:0=1111, then P0↑

Delay for N-bit adder (N even) is N+3  
(compare with 2N+1 for original circuit)

## Carry Lookahead (1)

For each bit of an N-bit adder we get a carry out ( $CO=1$ ) if two or more of  $P, Q, CI$  are equal to 1.

There are three possibilities:

- $P, Q=00$ :  $C=0$  always *Carry Inhibit*
- $P, Q=01$  or  $10$ :  $C=CI$  *Carry Propagate*
- $P, Q=11$ :  $C=1$  always *Carry Generate*

We define three signals:

- $CG = P \cdot Q$  *Carry Generate*
- $CP = P \oplus Q$  *Carry Propagate*
- $CGP = P + Q$  *Carry Generate or Propagate*

We get a carry out from a bit position either if that bit generates a carry ( $CG=1$ ) or else if it propagates the carry and there is a carry in from the previous bit ( $CP \cdot CI = 1$ ):

$$C = CG + CP \cdot CI$$

Since  $CGP = CG + CP$ , an alternate expression is:

$$C = CG + CGP \cdot CI$$

The second expression is usually used since  $P + Q$  is easier and faster to generate than  $P \oplus Q$ .



## Carry Lookahead (2)

Consider all the ways in which we get a carry out of bit position 3:

- |   |                  |
|---|------------------|
| 1) Bit 3 generates a carry:   | 1???             |
|   | + <u>1???</u>    |
|   |                  |
| 2) Bit 2 generates a carry <u>and</u><br>bit 3 propagates it.   | 11??             |
|   | + <u>01??</u>    |
|   |                  |
| 3) Bit 1 generates a carry <u>and</u><br>bit 2 propagates it <u>and</u><br>bit 3 propagates it.                                   | 101?             |
|   | + <u>011?</u>    |
|   |                  |
| 4) Bit 0 generates a carry <u>and</u><br>bit 1 propagates it <u>and</u><br>bit 2 propagates it <u>and</u><br>bit 3 propagates it. | 1011             |
|   | + <u>0101</u>    |
|   |                  |
| 5) The C-1 input is high <u>and</u><br>bits 0,1,2 and 3 all propagate the carry.  | 1011             |
|   | + <u>0100</u> +1 |

Thus

$$C_3 = C_G3 + C_P3 \cdot C_G2 + C_P3 \cdot C_P2 \cdot C_G1 + C_P3 \cdot C_P2 \cdot C_P1 \cdot C_G0 + C_P3 \cdot C_P2 \cdot C_P1 \cdot C_P0 \cdot C_{-1}$$

As before, we can use  $C_{GPn}$  in place of  $C_{Pn}$ .



## Carry Lookahead (4)

Carry lookahead circuit complexity for N-bit adder:

- Expression for  $C_n$  involves  $n+2$  product terms each containing an average of  $\frac{1}{2}(n+3)$  input signals.
- Direct implementation of equations for all  $N$  carry signals involves approx  $N^3/3$  transistors.

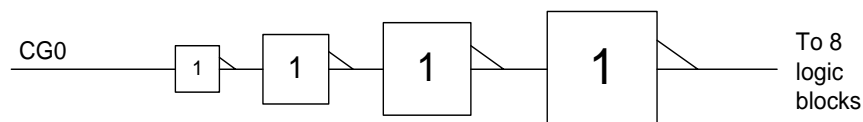
$$N = 64 \Rightarrow N^3/3 = 90,000$$

- By using a complex CMOS gate, we can actually generate  $C_n$  using only  $4n+6$  transistors so all  $N$  signals require approx  $2N^2$  transistors.

$$N = 64 \Rightarrow 2N^2 = 8,000$$

Actual gain is not as great as this because for large  $n$ , the expression for  $C_n$  is too big to use a single gate.

- $C_{-1}$ ,  $CG_0$  and  $CGP_0$  must drive  $N-1$  logic blocks. For large  $N$  we must use a chain of buffers to reduce delay:



The circuit delay is thus not quite independent of  $N$ .

## Quiz

1. What does it mean to say that a full-adder is *self-dual* ?
2. How does placing an inverter between each stage of a multi-bit adder allow the merging of gates in consecutive stages ?
3. In a 4-bit adder, give an example of a propagation delay that *increases* when alternate bits are inverted.
4. Why is a carry-lookahead adder generally implemented using CGP rather than CP outputs ?

## Lecture 15

**Fast Adder Circuits (2)**

## Objectives

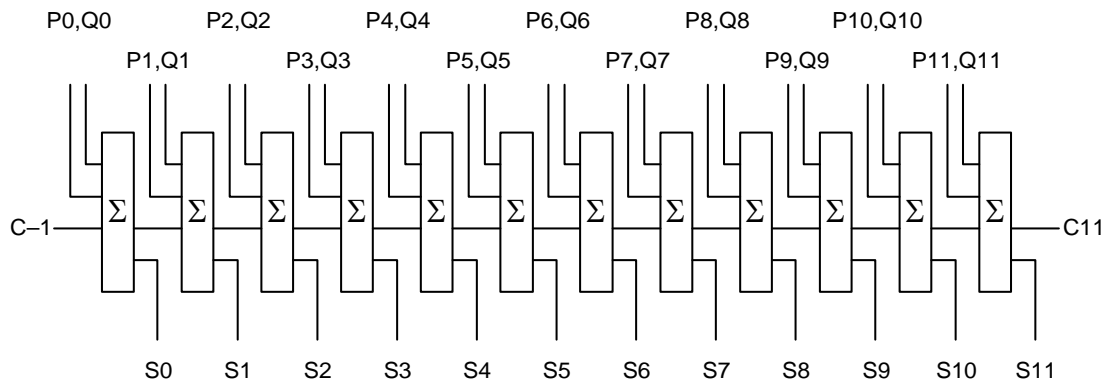
- Understand the *carry skip* technique for reducing the propagation delay of an adder circuit.
- Understand how the *carry save* technique can be used when adding together several numbers.

## Summary So Far:

- Cascading full adders:  
2N+1 gate delays, 50N transistors
- Use self-duality to invert odd-numbered stages:  
N+3 gate delays, 61N transistors
- Carry lookahead:  
6 gate delays, between  $2N^2$  and  $0.3N^3$  transistors

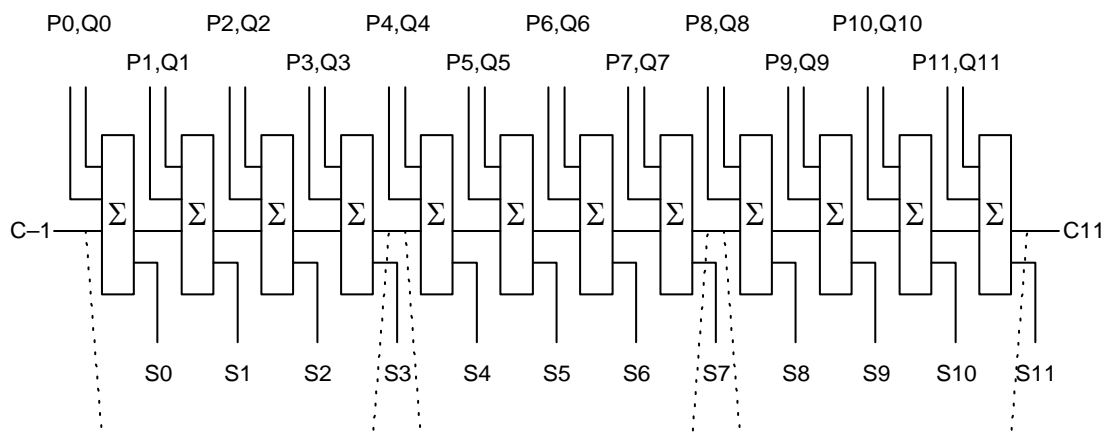
## Carry Skip (1)

Consider a 12-bit adder:



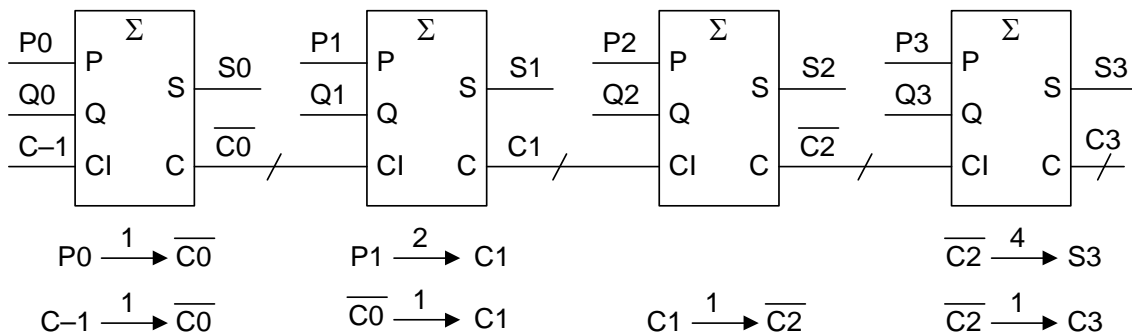
The worst-case delay path is from  $C_{-1}$  to  $S_{11}$ .

In *carry skip*, we speed up this path by allowing the carry signal to skip over several adder stages at a time:



## Carry Skip (2)

Consider our fast adder circuit without carry lookahead (but using alternate-bit inversion):



There are two possible sorts of addition sum:

- All bits propagate the carry  $\Rightarrow C_3 = C_{-1}$ :

$$\begin{array}{r}
 0101 \\
 1010 \\
 \underline{\phantom{0}0} \\
 01111
 \end{array}
 \qquad
 \begin{array}{r}
 0101 \\
 1010 \\
 \underline{\phantom{0}1} \\
 10000
 \end{array}$$

$$C_{-1} \uparrow \rightarrow C_3 \uparrow = 4 \text{ gate delays}$$

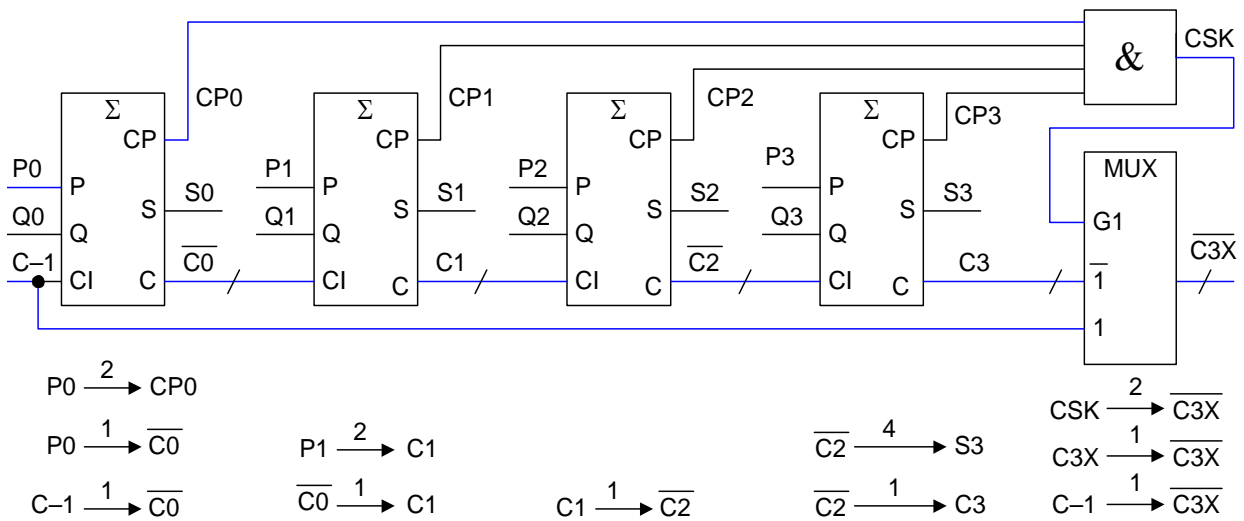
- At least one bit doesn't propagate the carry  $\Rightarrow C_3$  is completely independent of  $C_{-1}$ :

$$\begin{array}{r}
 0101 \\
 1110 \\
 \underline{\phantom{0}0} \\
 10011
 \end{array}
 \qquad
 \begin{array}{r}
 0101 \\
 1110 \\
 \underline{\phantom{0}1} \\
 10100
 \end{array}$$

$$C_{-1} \uparrow \rightarrow C_3 = 0 \text{ gate delays}$$

### Carry Skip (3)

We speed up  $C-1 \rightarrow C3$  by detecting when all bits propagate the carry and using a multiplexer to allow  $C-1$  to skip all the way to  $C3$ :



Calculate Carry Propagate ( $CP = P \oplus Q$ ) for each bit. Call this 2 gate delays since XOR gates are slow.  $CSK=1$  if all bits propagate the carry.

- **Case 1:** All bits propagate the carry  
 $C-1 \rightarrow \overline{C3X} = 1$  gate delay (via multiplexer)
- **Case 2:** At least one bit inhibits or propagates the carry  
 $\Rightarrow$  **C-1 does not affect C3**

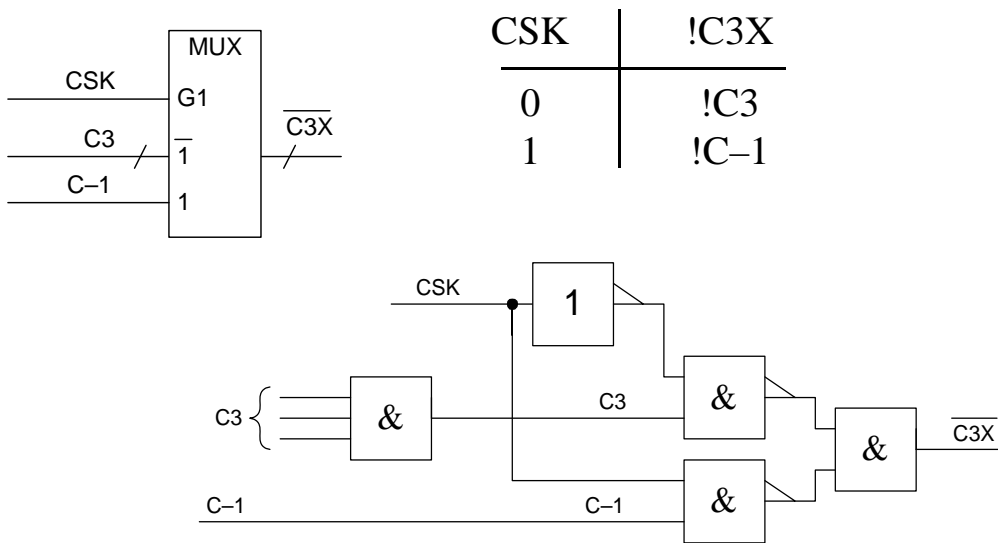
Longest delays to  $\overline{C3X}$  and  $S3$ :

- $P0 \rightarrow \overline{C3X} = 5$  (via either  $\overline{C0}$  or  $CSK$ )
- $P0 \rightarrow S3 = 7$



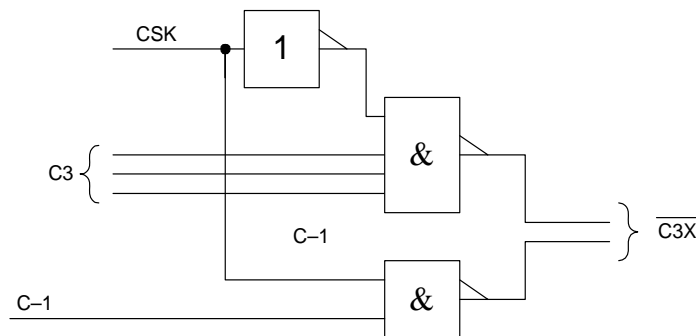
## Carry Skip (4)

### Multiplexer Details



We merge both AND gates:

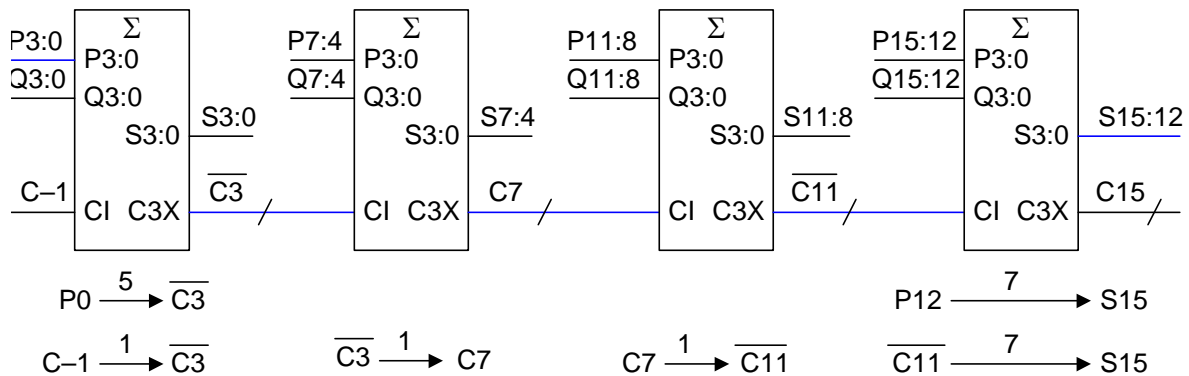
- the 3-AND gate merges into the following NAND
- the 2-AND gate merges into the next adder stage



C-1 → !C3X now equals 1 gate delay.

## Carry Skip (5)

Combine 4 blocks to make a 16-bit adder:



Worst-case delay is:

$$P0 \rightarrow !C3 \rightarrow C7 \rightarrow !C11 \rightarrow S15 = 14 \text{ gate delays}$$

Each additional block of 4 bits gives a delay of only 1 gate delay: this corresponds to  $\frac{1}{4}$  gate delay per bit.

For an N-bit adder we have a delay of  $\frac{1}{4}N+10$ . We can reduce this still further by having larger super-blocks.

### Carry circuit delays:

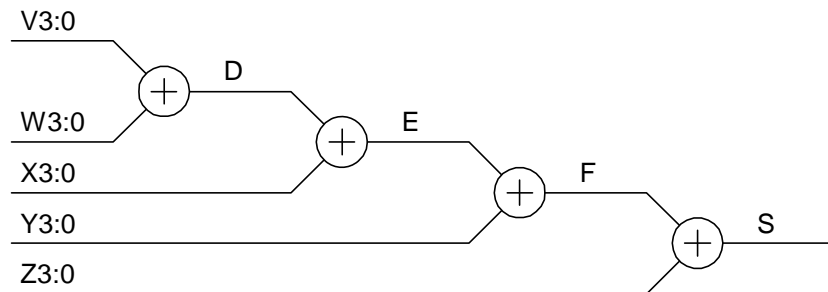
Simple	$2N+1$
Bit-inversion	$N+3$
Carry Skip	$\frac{1}{4}N+10$
Carry Lookahead	6

- but lots of circuitry and high gate fanout  $\Rightarrow$  more delay

## Adding lots of numbers

In multiplication circuits and digital filters we need to add lots of numbers together.

Suppose we want to add together five four-bit unsigned numbers: V, W, X, Y and Z.



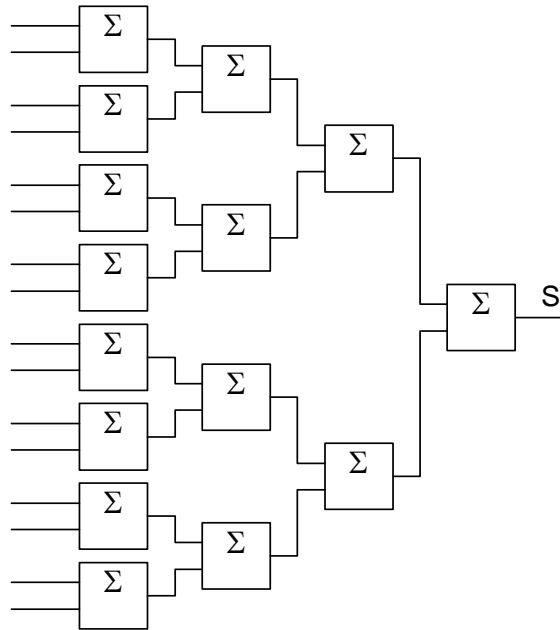
If we use carry-lookahead adders, each stage will have 6 gate delays.

Total delay to add together K values will be  $(K-1) \times 6$ .

Thus  $K=16$  gives a delay of 90 gate delays.

## Addition Tree

In practice we use a tree arrangement of adders:



Number of values, K	16	8	4	2	1
$\log_2(K)$	4	3	2	1	0

Each column of adders adds a delay of 6 and halves the number of values needing to be added together.

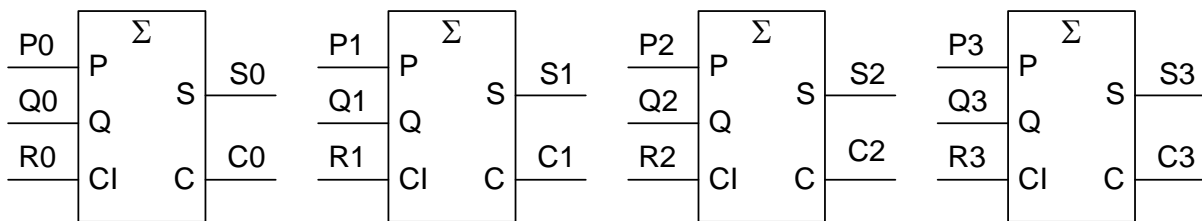
Equivalently, each column of adders reduces  $\log_2 K$  by one.

Hence the total delay is  $\log_2 K \times 6$  giving a delay of 24 to add together 16 values.

The total number of adders required is still  $K-1$  as before.

## Carry-Save Adder

Take a normal 4-bit adder but *don't connect up the carries*:



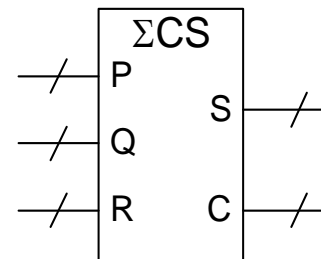
We have  $P+Q+R = 2C + S$

E.g.  $P=9, Q=12, R=13$   
gives  $C=13, S=8$

P:	1001
Q:	1100
R:	1101
<hr style="border: 0.5px solid black;"/>	
S:	1000
C:	1101_

We call this a *carry-save* adder: it reduces the addition of 3 numbers to the addition of 2 numbers.

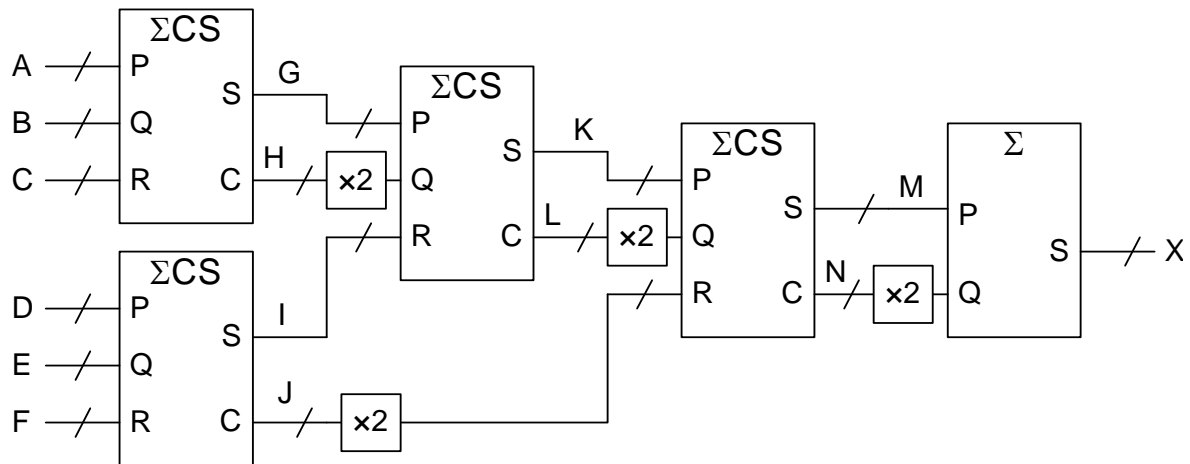
The propagation delay is 3 gates regardless of the number of bits. The amount of circuitry is much less than a carry-lookahead adder.



The circuit reduces  $\log_2 K$  by 0.585 (from 1.585 to 1.0) for a delay of 3. The overall delay we can expect is therefore  $\log_2 K \times 3/0.585 = \log_2 K \times 5.13$ . This is *better* than carry lookahead for *less circuitry*.

## Carry Save Example

We will calculate:  $13+10+5+11+12+1 = 52$

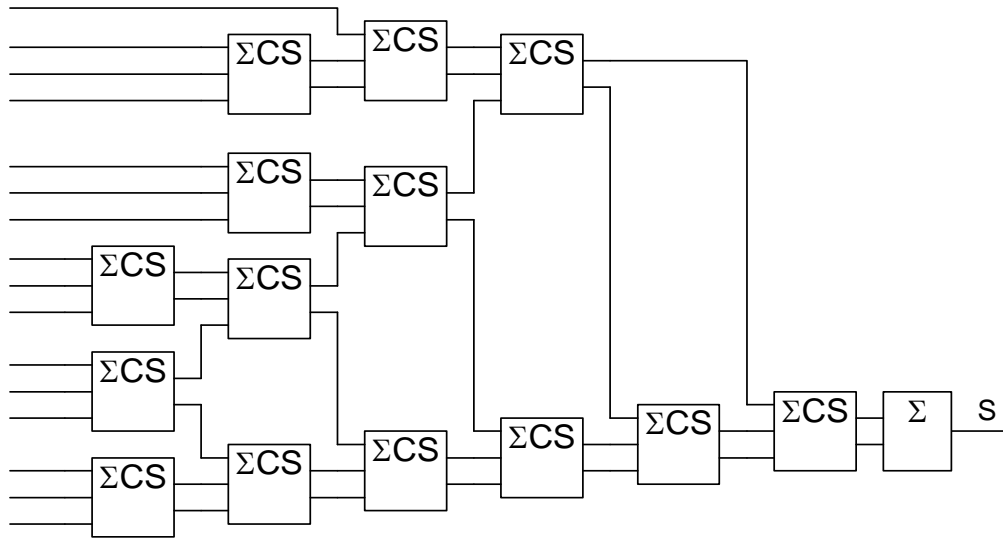


A: 1101	G: _0010	
B: 1010	2H: 1101_	
C: 0101	I: _0110	
<u>  </u>	<u>  </u>	
G: 0010	K: 11110	M: 01000
H: 1101_	L: _0010_	2N: 1011__
		<u>  </u>
		X: 0110100
D: 1011	K: 11110	
E: 1100	2L: 0010_	
F: 0001	2J: 1001_	
<u>  </u>	<u>  </u>	
I: 0110	M: 01000	
J: 1001_	N: 1011__	

- Notes:**
1. x2 requires no logic: just connect wires appropriately
  2. No logic required for adder columns with only 1 input
  3. All adders are actually only 4 bits wide
  4. Final addition M+2N requires a proper adder

## Carry-Save Tree

We can construct a tree to add sixteen values together:



Number of values, K	16	13	9	6	4	3	2	1
$\log_2(K)$	4	3.7	3.17	2.58	2	1.58	1	0
Delay	0	3	6	9	12	15	18	24
$\Delta\text{Delay}/\Delta\log_2(K)$	10	5.65	5.13	5.13	7.23	5.13	6	

- The final stage must be a normal adder because we need to obtain a single output.
- The delay is the same as for a conventional lookahead-adder tree but uses much less circuitry.
- The irregularity of the tree causes a reduction in efficiency but this is relatively small (and becomes even smaller for large K).
- Inverting alternate stages will speed up both tree circuits still further but requires more circuitry.

*Merry Christmas*



*The End*



## Quiz

1. In a 4-bit adder, how can you tell from  $P_{0:3}$  and  $Q_{0:3}$  whether or not  $C_3$  is dependent on  $C_{-1}$  ?
2. A multiplexer normally has 2 gate delays from its data inputs to its output. How is this reduced to 1 gate delay in the carry skip circuit ?
3. If five 4-bit numbers are added together, how many bits are needed to represent the result ?