

Lecture 16

Digital Number Systems
Boolean Logic
Truth Tables
Basic Logic Gates

Positional Number Systems

- What does 5132.13 really mean?
- Depends on the number base!
- Assuming base 10 (decimal): Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

$$5132.13_{10} = 5 \times 10^3 + 1 \times 10^2 + 3 \times 10^1 + 2 \times 10^0 + 1 \times 10^{-1} + 3 \times 10^{-2}$$

- Assuming base 6: Digits: 0, 1, 2, 3, 4, 5

$$5132.13_6 = 5 \times 6^3 + 1 \times 6^2 + 3 \times 6^1 + 2 \times 6^0 + 1 \times 6^{-1} + 3 \times 6^{-2}$$

- Base 2 (binary): Digits: 0, 1

$$1011.11_2 = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})$$

Binary Vocabulary

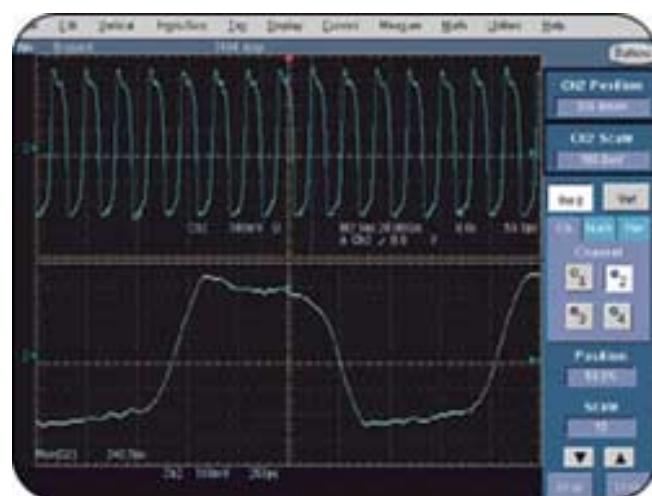
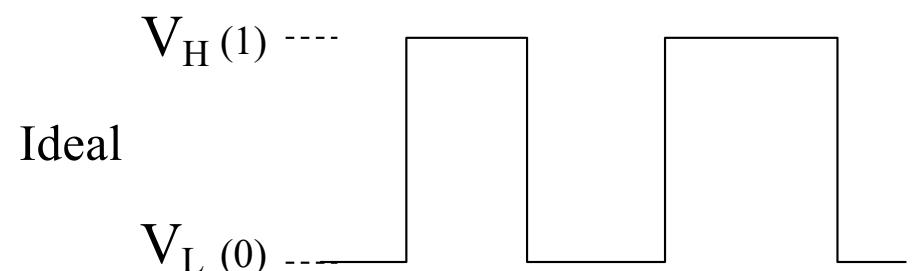
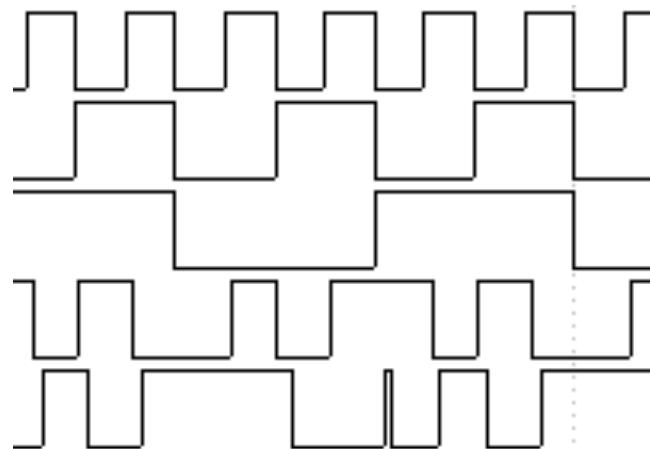
- A single Binary Digit = Bit. Either a 0 or 1
- By using groups of bits, we can achieve higher precision.
 - 8 bits => each bit pattern represents 1/256; 1 out of 2^8
 - 16 bits => each bit pattern represents 1/65,536; 1 out of 2^{16}
 - 32 bits => each bit pattern represents 1/4,294,967,296; 1 out of 2^{32}
 - 64 bits => each bit pattern represents 1/18,446,744,073,709,550,000; 1 out of 2^{64}
- LSB (least significant bit): The rightmost bit, weight = $2^0 = 1$
- MSB (most significant bit): The leftmost bit, weight = 2^{n-1}
- Data sizes
 - 1 Nibble (or nybble) = 4 bits
 - 1 Byte = 2 nibbles = 8 bits
 - 1 Kilobyte (KB) = 1024 bytes
 - 1 Megabyte (MB) = 1024 kilobytes = 1,048,576 bytes
 - 1 Gigabyte (GB) = 1024 megabytes = 1,073,741,824 bytes

Representation of Digital Signals

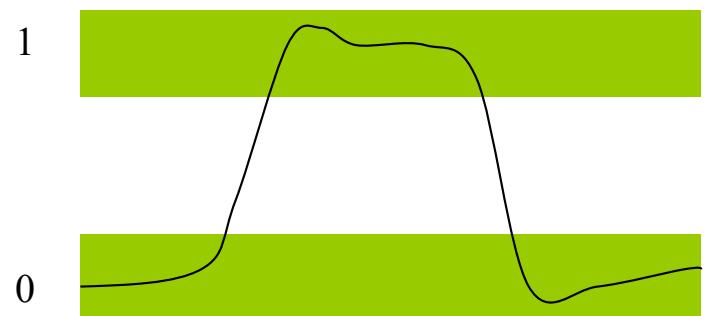
Technology	Representing 0	Representing 1
Pneumatic Logic	Fluid at low pressure	Fluid at high pressure
Relay Logic	Circuit open	Circuit closed
CMOS	0 - 1.5 V	3.5 - 5.0 V
TTL	0 - 0.8 V	2.0 - 5.0 V
Fiber Optics	Light off	Light on
Dynamic Memory	Capacitor discharged	Capacitor charged
Nonvolatile, erasable memory	Electrons trapped	Electrons released
Bipolar ROM	Fuse blown	Fuse intact
Bubble memory	No magnetic bubble	Magnetic bubble
Magnetic tape or disk	Flux direction “north”	Flux direction “south”
Polymer memory	Molecule in State A	Molecule in State B
Read-only Compact Disk	No pit	Pit
Re-writeable compact disk	Dye in crystalline state	Dye in non-crystalline state
Quantum Bit (qbit)	Positive Spin	Negative Spin

Anything that has two states can be used to represent a digital signal.

Digital Signals



Actual



What Kinds of Data Can Be Represented in Binary?

- Data with discrete values, or data that can be approximated with discrete values
 - **Numbers** – signed, unsigned, integers, floating point, complex, rational, irrational, ...
 - **Text** – characters, strings, ...
 - **Images** – pixels, colors, shapes, ...
 - **Sound** – pitch, amplitude, ...
 - **Logical** – true / false, open / closed, on / off, ...
 - **Instructions** – programs, ...
 - ...

Conversion from Binary to Decimal

Convert 101011.11_2 to base 10:

$$= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$

$$= 32 + 0 + 8 + 0 + 2 + 1 + \frac{1}{2} + \frac{1}{4}$$

$$= 43.75_{10}$$

Convert 114_{10} to binary

$$\begin{array}{r} 114 \\ - 64 \\ \hline 50 \\ - 32 \\ \hline 18 \\ - 16 \\ \hline 2 \\ - 0 \\ \hline 2 \\ - 0 \\ \hline 2 \\ - 2 \\ \hline 0 \\ - 0 \\ \hline 0 \end{array} \quad \begin{array}{l} 1 \times 2^6 \\ 1 \times 2^5 \\ 1 \times 2^4 \\ 0 \times 2^3 \\ 0 \times 2^2 \\ 1 \times 2^1 \\ 0 \times 2^0 \end{array}$$

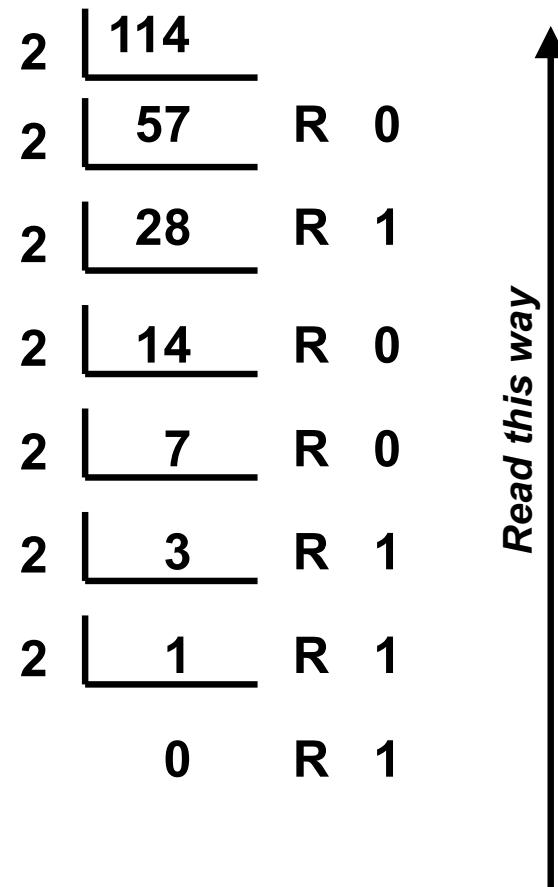
Read this way



$$114_{10} = 1110010_2$$

This method also works for fractional numbers.

An Alternate Method



$$114_{10} = 1110010_2$$

Converting Fractions from base 10 to binary:

Convert 0.7_{10} to binary

Read this way

$$\begin{array}{r} 0.7 \\ \times 2 \\ \hline (1).4 \\ \times 2 \\ \hline (0).8 \\ \times 2 \\ \hline (1).6 \\ \times 2 \\ \hline (1).2 \\ \times 2 \\ \hline (0).4 \\ \times 2 \\ \hline (0).8 \end{array}$$

$0.7_{10} = 0.1 \underline{0110} \underline{0110} \dots_2$

process starts repeating here

Hexadecimal Notation

- Binary is hard to read and write by hand
- Hexadecimal (base 16) is a common alternative
 - 16 digits are 0123456789ABCDEF

0100	0111	1000	1111	=	0x478F
1101	1110	1010	1101	=	0xDEAD
1011	1110	1110	1111	=	0xBEEF
1010	0101	1010	0101	=	0xA5A5

1. Separate binary code into groups of 4 bits (starting from the right)
2. Translate each group into a single hex digit

0x is a common prefix for writing numbers which means hexadecimal

Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Binary Codes for Characters

ASCII Code

- *American Standard Code for Information Interchange*
- 7-bit
- letters
- symbols
- terminal codes
- extend to byte

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	Ø	96	60	140	`	'
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Example: encode text with ASCII

Text to encode: **Hi there!**

Letter	ASCII decimal	Hexadecimal	Binary
H	72	48	0100 1000
i	105	69	0110 1001
	32	20	0010 0000
t	116	74	0111 0100
h	104	68	0110 1000
e	101	65	0110 0101
r	114	72	0111 0010
e	101	65	0110 0101
!	33	21	0010 0001

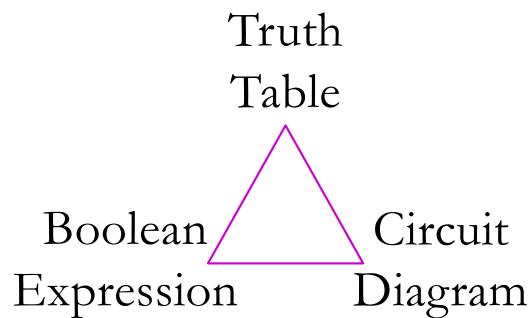
Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Letter:	H	i	<space>	t	h	e	r	e	!
Hex:	48	69	20	74	68	65	72	65	21
Binary:	01001000	011010001	00100000	01110100	01101000	01100101	01110010	01100101	00100001

Binary Logic Functions

- Binary digits 1 and 0 can be used to represent Logical values True and False.
- Binary can implement Logic Functions.
- Three representations for logic functions:

- Truth Tables
- Boolean Equations
- Logic Gates



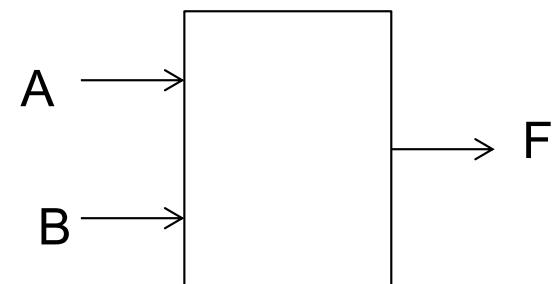
- Learn to convert from one format to another

Truth Tables

- A truth table provides a complete enumeration of the inputs and the corresponding output for a function.

If there are n inputs,
There will be 2^n rows
In the table.

A	B	F
0	0	1
0	1	1
1	0	0
1	1	1



Unlike with regular algebra, full enumeration is possible
(and useful) in Boolean Algebra

Inversion Operation

Also known as *complementation* or *not*.

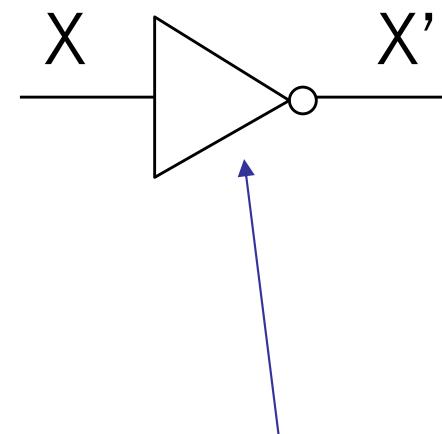
The inverse of x is written as x' or \bar{x} in Boolean Algebra.

$$Q = X'$$

or

$$Q = \bar{X}$$

X	Q
0	1
1	0



This is a *schematic* symbol.
It is a graphical representation
of a circuit which implements
the inversion operation.

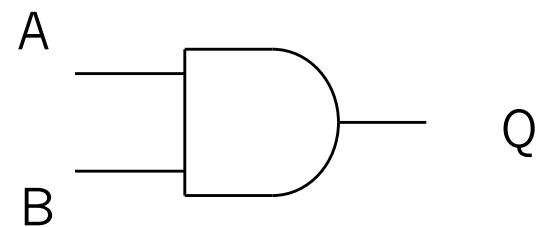
AND Operation

Output is true iff *all* inputs are true

“•” denotes AND operation in Boolean Algebra

$$Q = A \bullet B$$

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

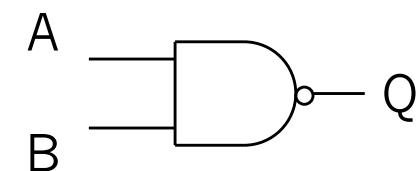


NAND Operation

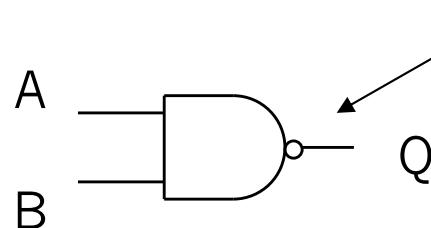
$$Q = (A \cdot B)'$$

$$\underline{Q = A \cdot B}$$

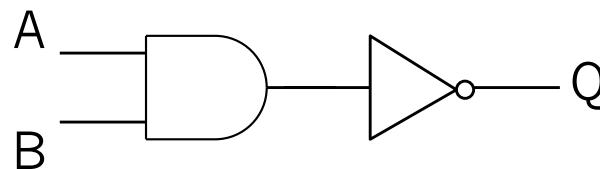
A	B	$Q = (A \cdot B)'$
0	0	1
0	1	1
1	0	1
1	1	0



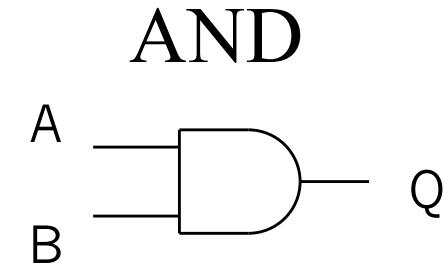
Q is false iff A **AND** B are true



Bubble means *NOT*



Equivalent Logic Circuit



Q is true iff A **AND** B are true

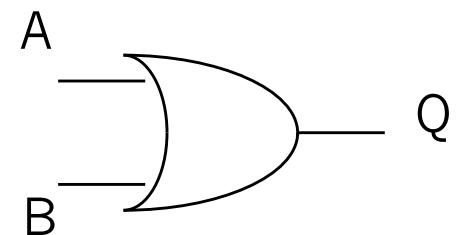
OR Operation

Output is true if *any* inputs are true

“+” denotes OR operation in Boolean Algebra

$$Q = A + B$$

A	B	$Q = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

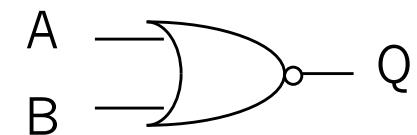


NOR Operation

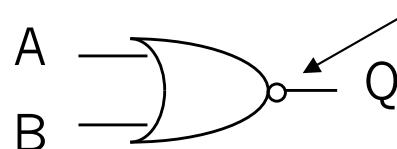
$$Q = (A + B)'$$

$$Q = A + B$$

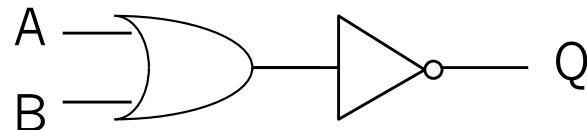
A	B	$Q = (A+B)'$
0	0	1
0	1	0
1	0	0
1	1	0



Q is false iff A **OR** B is true

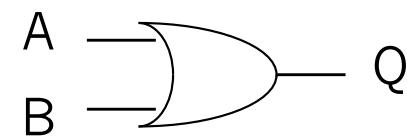


Bubble means *NOT*



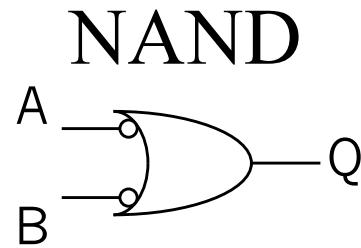
Equivalent Logic Circuit

OR



Q is true iff A **OR** B is true

Alternative Gate Symbols

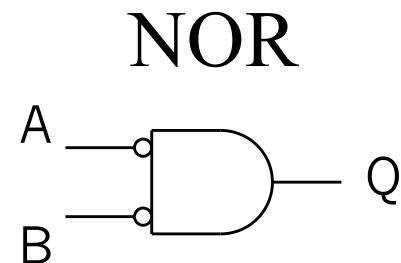


Q is true iff A is false OR B is false

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NAND

An alternative NAND gate symbol where the inputs A and B are shown as a single horizontal line, and the output Q is shown as a small circle at the end of the line.



Q is true iff A is false AND B is false

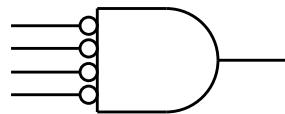
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

NOR

An alternative NOR gate symbol where the inputs A and B are shown as a single horizontal line, and the output Q is shown as a small circle at the end of the line.

Example Use

- Design a circuit to determine whether the bits of a 4-bit wire are all zero



This is the appropriate NOR gate symbol to use...

AND/OR to NAND/NAND

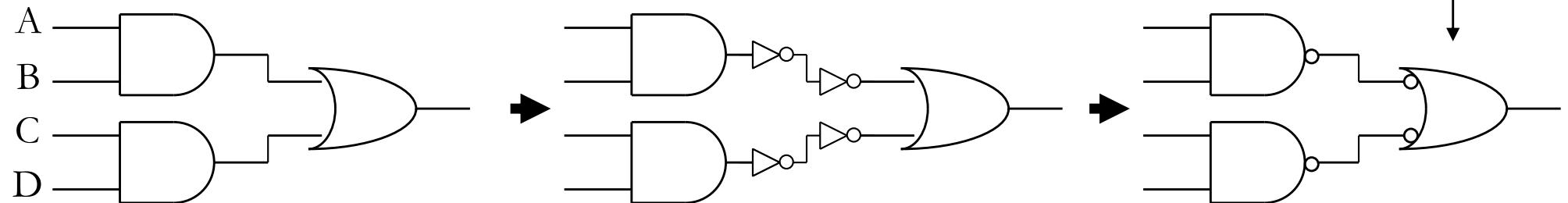
Algebra-based:

$$\begin{aligned} AB + CD &= [AB + CD]'' \\ &= [(AB)'(CD)']' \end{aligned}$$

NAND gates are preferred because we can make them with CMOS transistors

Schematic-based:

Preferred symbol in this context...



OR/AND to NOR/NOR

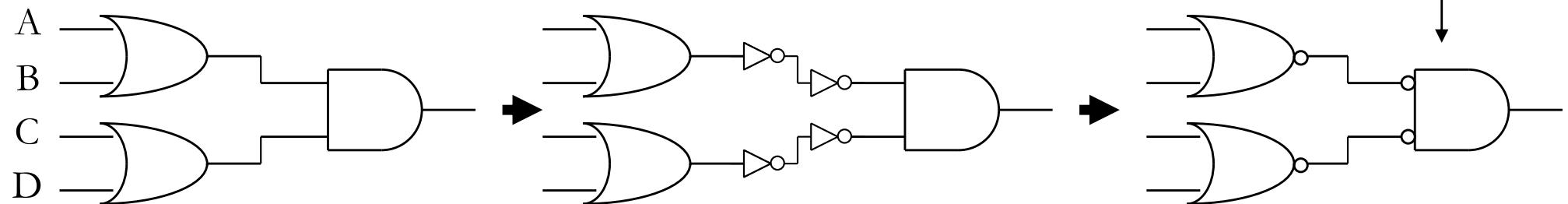
Algebra-based:

$$\begin{aligned}(A+B)(C+D) &= [(A+B)(C+D)]'' \\ &= [(A+B)' + (C+D)']'\end{aligned}$$

NOR gates are preferred because we can make them with CMOS transistors

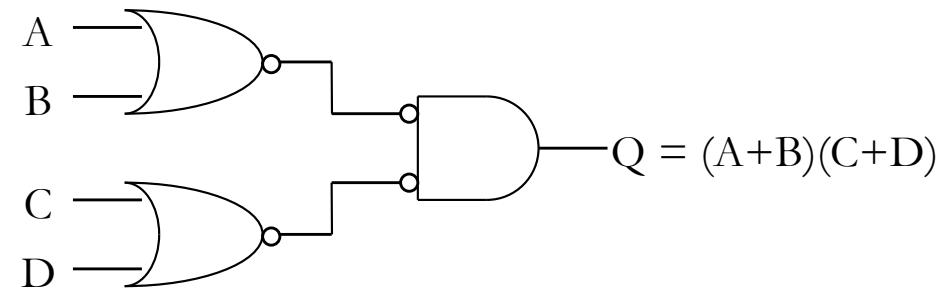
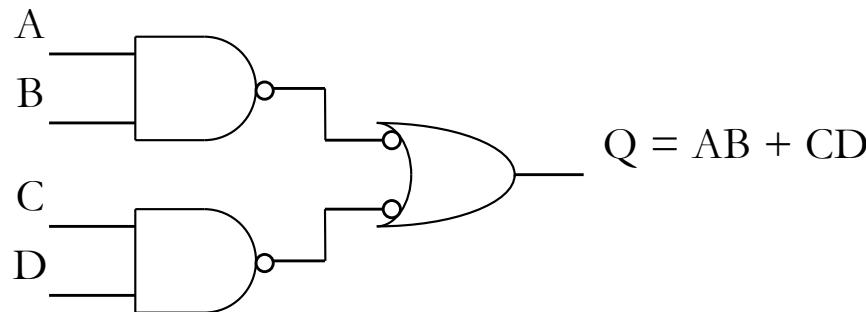
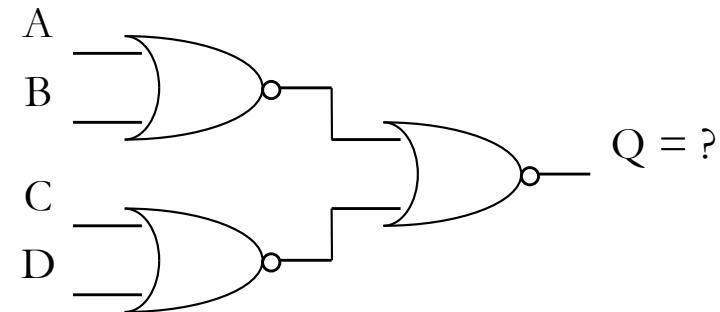
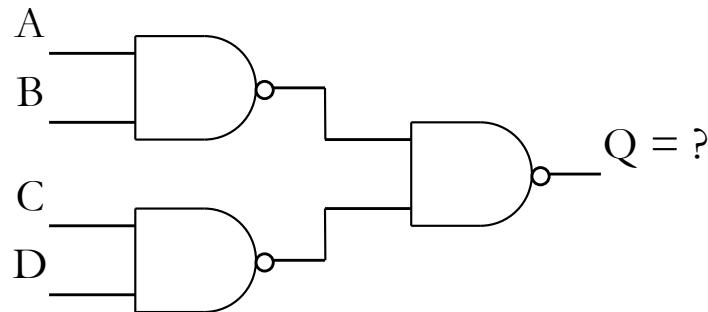
Schematic-based:

Preferred symbol
in this context...



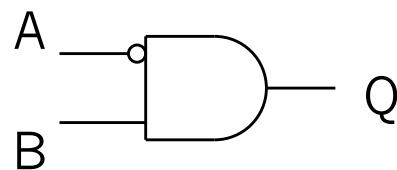
Using Alternative Gate Symbols

Which is easier to understand?



You can think of the bubbles as canceling each other out...

Mixed Symbols



Q is true iff A is false AND B is true

- Such a gate doesn't have a name
- Implement using an AND gate and an inverter
- Simplifies schematics, enhances readability

Exclusive-OR (XOR)

Output is true iff inputs are not the same

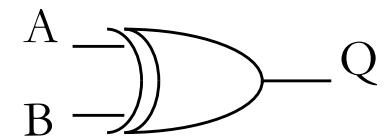
\oplus is the symbol for the XOR operator in Boolean Algebra

$$Q = A \oplus B$$

$$Q = A'B + AB'$$

$$Q = \overline{A}B + A\overline{B}$$

A	B	$Q = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



Another definition:

Q is true iff A does not equal B

Exclusive-OR Theorems

$$X \oplus 0 = X$$

$$X \oplus 1 = X'$$

$$X \oplus X = 0$$

$$X \oplus X' = 1$$

$$X \oplus Y = Y \oplus X$$

Commutative law

$$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$$

Associative law

$$(X \oplus Y)' = X \oplus Y' = X' \oplus Y$$

Equivalence Operation

Output is true iff inputs are equal

\equiv denotes equivalence operator

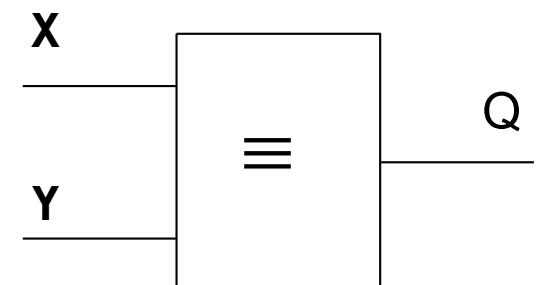
(also written as $X==Y$) in Boolean Algebra

$$Q = (X == Y)$$

$$Q = X'Y' + XY$$

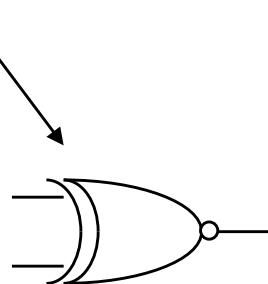
$$Q = \overline{X}\overline{Y} + XY$$

X	Y	$Q = (X == Y)$
0	0	1
0	1	0
1	0	0
1	1	1



XOR and EQUIV are Complements !!

Alternate equivalence symbol



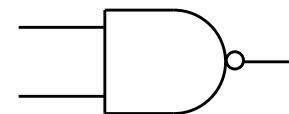
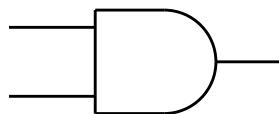
X	Y	$X \oplus Y$	$X == Y$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Gate often called *exclusive NOR* or *XNOR*

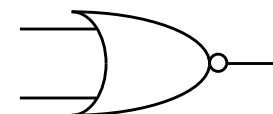
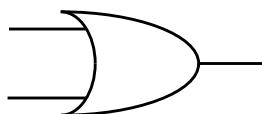
Types of Gates

- Gates already studied

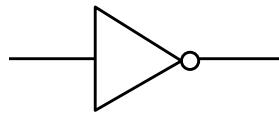
- AND, NAND



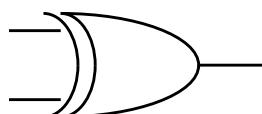
- OR, NOR



- Inverter (NOT)



- XOR (Exclusive-OR)



- XNOR (Equivalence)



Boolean Algebra

Boolean expressions are made up of variables and constants combined by AND, OR and NOT (and sometimes XOR and XNOR)

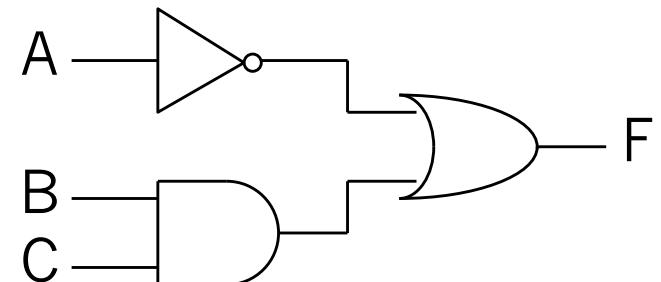
Examples: 1 A' $A \cdot B$ $C + D$ AB $A(B+C)$ $AB' + C$

- $A \cdot B$ is the same as AB (\cdot is omitted when obvious).
- Parentheses are used like in regular algebra for grouping.
- **NOT** has highest precedence, followed by **AND** then **OR**.

A **term** is a grouping of variables ANDed together.

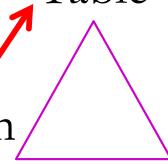
Each Boolean expression has a corresponding implementation with logic gates.

$$F = A' + BC$$



Boolean Expressions

Each Boolean expression can be specified by a truth table which lists all possible combinations of the values of all variables in the expression.

Boolean Expression  Circuit Diagram

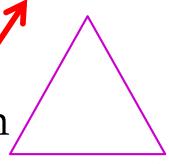
Truth Table

$$F = A' + BC$$

A	B	C	A'	BC	F = A' + BC
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

Converting Boolean Functions to Truth Tables

$$F = AB + BC$$

Boolean Expression  Circuit Diagram

Truth Table

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
BC		0 1 1	1
BC		1 0 0	0
BC		1 0 1	0
BC		1 1 0	1 } AB
BC		1 1 1	1 }

Boolean Expressions From Truth Tables

Each 1 in the output of a truth table specifies one term in the corresponding Boolean expression.

The expression can be read off by inspection...

Boolean Expression Diagram

Truth Table

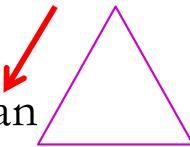
A	B	C	F	F is true when:
0	0	0	0	
0	0	1	0	
0	1	0	1	A is false AND B is true AND C is false
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	A is true AND B is true AND C is true

OR

$F = A' BC' + ABC$

Another Example

Boolean Expression Truth Table
Circuit Diagram



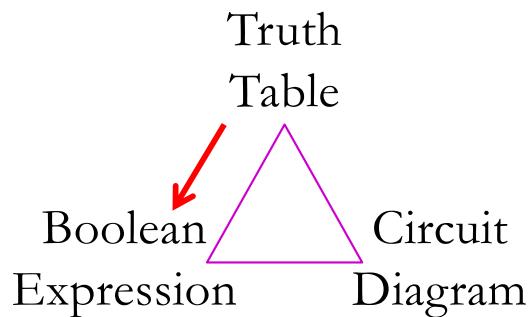
A	B	C		F
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		0
1	0	0		1
1	0	1		0
1	1	0		0
1	1	1		1

F = ?

$$\begin{aligned} F = & A' B' C + \\ & A' B C' + \\ & A B' C' + \\ & A B C \end{aligned}$$

These 4 terms are called **minterms** because they are terms that contain all three literals: A, B, and C. Minterms always map to a single row in the Truth Table.

Yet Another Example



A	B	F
0	0	1
0	1	1
1	0	1
1	1	1

$F = ?$

$$\begin{aligned} F &= A'B' + A'B + AB' + AB \quad (\text{minterms}) \\ &= A'(B'+B) + A(B'+B) \\ &= A' + A \\ &= 1 \end{aligned}$$

(F is always true)

There may be multiple expressions for any given truth table.

Basic Boolean Algebra Theorems

$$X + 0 = X$$

$$X + 1 = 1$$

$$X + X = X$$

$$X + X' = 1$$

$$(X')' = X$$

$$X \cdot 1 = X$$

$$X \cdot 0 = 0$$

$$X \cdot X = X$$

$$X \cdot X' = 0$$

Basic Boolean Algebra Theorems

While these laws don't seem very exciting, they can be very useful in simplifying Boolean expressions:

Simplify:

$$\underbrace{(M N' + M' N) P + P' + 1}_{\begin{array}{c} X \\ + 1 \end{array}} = 1$$

Commutative Laws

$$X \cdot Y = Y \cdot X \quad X + Y = Y + X$$

Associative Laws

$$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$$

$$(X + Y) + Z = X + (Y + Z) = X + Y + Z$$

Just like regular algebra

Boolean Proof

Using Truth Tables

Truth Tables can be used to prove that 2 Boolean expressions are equal.

If the two expressions have the same value for all possible combinations of input variables, they are equal.

$$X Y' + Y = X + Y$$

X	Y	Y'	X Y'	X Y' + Y	X + Y
0	0	1	0	0	0
0	1	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

=

These two expressions are equal.

Distributive Law

$$X(Y + Z) = XY + XZ$$

Prove with a truth table:

X	Y	Z	Y+Z	X(Y+Z)	XY	XZ	XY+XZ
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Just like regular algebra

=

Rules for Inversion

“DeMorgan’s Laws”

$$(X + Y)' = X' Y'$$

$$(XY)' = X' + Y'$$

$$X + Y = (X' Y')'$$

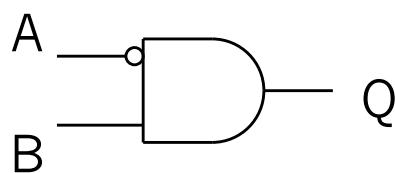
$$XY = (X' + Y')'$$

Proof:

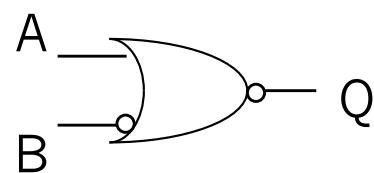
X	Y	X'	Y'	X + Y	(X + Y)'	X'Y'	X Y	(XY)'	X' + Y'
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0

Single Gate Conversion Rules

- Application of DeMorgan's Theorems to Gates:
 1. Change symbol
 - AND to OR
 - OR to AND
 2. Invert all inputs and outputs
- No change in behavior – merely a symbol change



equivalent to

Q is true
iff A is false AND B is true

Q is false if
A is true OR B is false

Functional Completeness

The set {AND, OR, NOT} is functionally complete

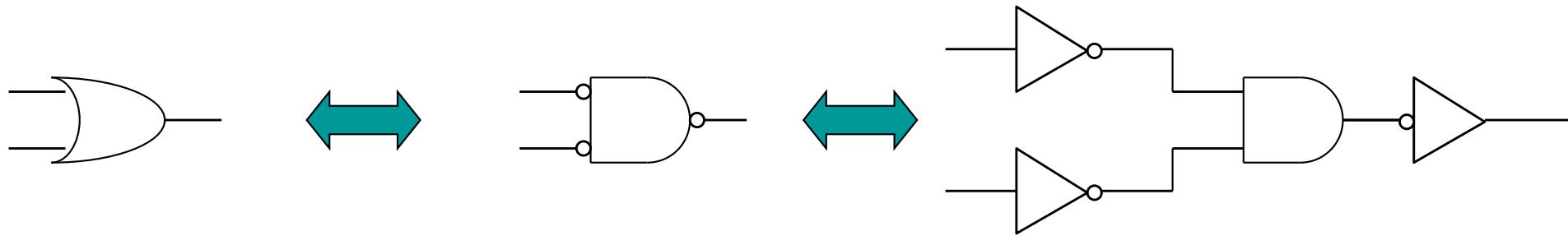
- All truth tables can be implemented using AND, OR, NOT gates
- All Boolean expressions can be written in terms of AND, OR, and NOT operators.

Other sets of operations may be functionally complete

- Any set of gates which can implement AND, OR and NOT is also functionally complete

Functionally Complete

- Is the set {AND, NOT} functionally complete?
- If I could just build an OR gate...



Success!

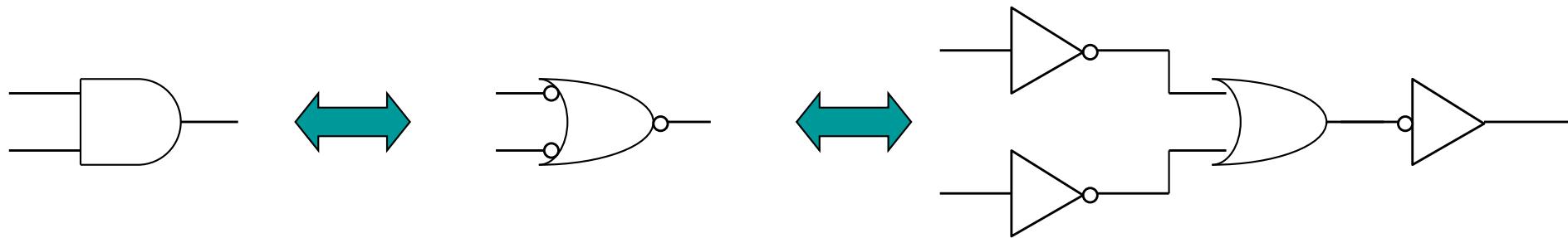
or...

$$X + Y = (X'Y')'$$

The set {AND, NOT} is functionally complete

Functionally Complete

- Is the set {OR, NOT} functionally complete?
- If I could just build an AND gate...



Success!

or...

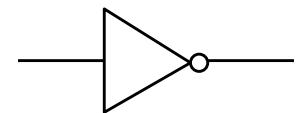
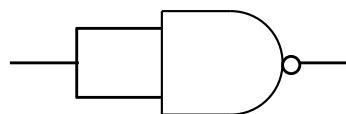
$$XY = (X' + Y')'$$

Functionally Complete

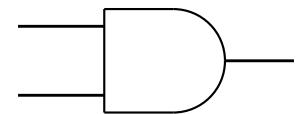
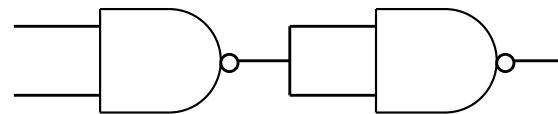
- Is the set {AND, OR} functionally complete?
- No! Can't build a NOT from set {AND, OR}

How About NAND Only?

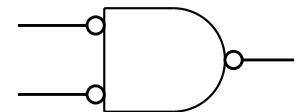
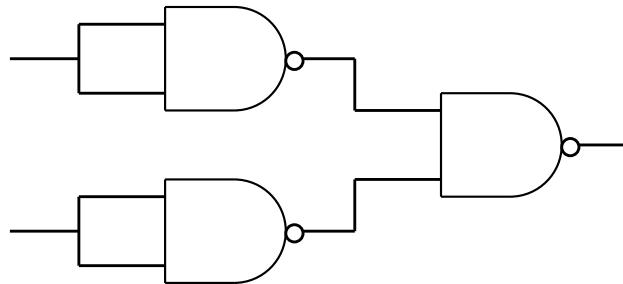
NOT



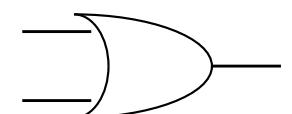
AND



OR

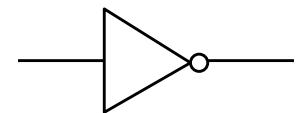
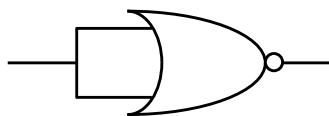


Success!

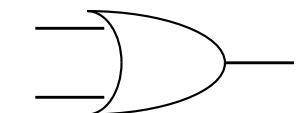
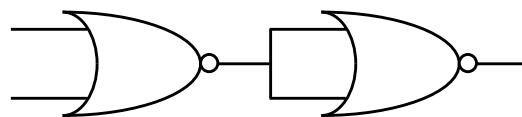


How About NOR Only?

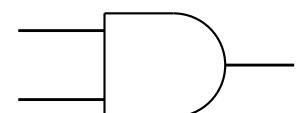
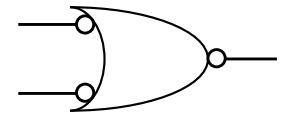
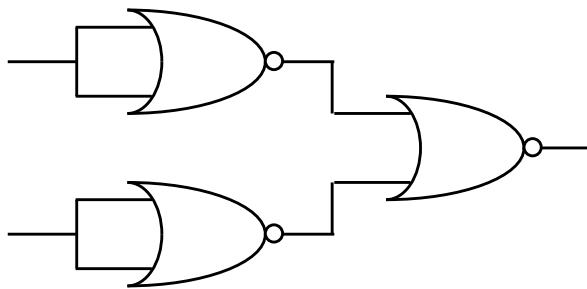
NOT



OR



AND



Success!

Functional Completeness

- A few functionally complete sets:

AND, OR, NOT
OR, NOT
AND, NOT
NOR
NAND

Lecture 17

Binary Arithmetic
Negative Numbers in Binary
Negation

Bitwise Logic Operations on Binary Words

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

A	NOT
0	1
1	0

a = 001100101
b = 110010100

a AND b = ?
a OR b = ?
NOT a = ?
a XOR b = ?



a = 001100101
b = 110010100

a & b	= 000000100
a b	= 111110101
~a	= 110011010
a ^ b	= 111110001

This is C-language syntax.

Binary Addition

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$1 + 1 = 0$ and carry 1 to the next column

Binary	Dec
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Examples:

$$\begin{array}{r} 0101 \\ + 0010 \\ \hline 0111 \end{array} \quad \begin{array}{l} (5_{10}) \\ (2_{10}) \\ (7_{10}) \end{array}$$

$$\begin{array}{r} 0101 \\ + 0011 \\ \hline 1000 \end{array} \quad \begin{array}{l} (5_{10}) \\ (3_{10}) \\ (8_{10}) \end{array}$$

↑↑↑ ← Carries

Binary Addition w/Overflow

Add 6-bit numbers 45_{10} and 44_{10} in binary

$$\begin{array}{rcl} & \begin{matrix} 1 & 1 & 1 \end{matrix} & \xleftarrow{\hspace{1cm}} \\ & 101101 & (45_{10}) \\ + & 101100 & (44_{10}) \\ \hline & 1011001 & (89_{10}) \end{array} \quad \text{Carries}$$

If the operands are unsigned, you can use the final carry-out as the MSB of the result.

Adding 2 k -bit numbers $\Leftrightarrow k+1$ bit result

More Binary Addition w/Overflow

Binary	Dec
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

$$\begin{array}{r} \text{1 1 1 1} \\ 1111 \quad (15_{10}) \\ +0001 \quad (1_{10}) \\ \hline 10000 \quad (16_{10}) \text{ 5 bit } (0_{10}) \text{ 4 bit} \end{array}$$

If you don't want a 5-bit result, just keep the lower 4 bits.

Here, 4-bits is insufficient to represent the result (16).

It *overflows* and wraps around back to 0.

Overflow – When the result is outside of the range of the number representation.

Signed Numbers: Two's Complement Representation

- Weight positional digits differently

$$0111_{2C} = 0 \times (-2^3) + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7_{10}$$

$$1111_{2C} = 1 \times (-2^3) + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = -1_{10}$$


Most significant bit (MSB) is given a negative weight...

Other bits have the same weights as unsigned numbers

The MSB of a 2's-complement number is called the 'Sign Bit'.
If Sign='1', then the number is negative. '0' means positive.

Unsigned Binary

2^2	2^1	2^0	Decimal
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

2's-Complement Binary

-2^2	2^1	2^0	Decimal
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

Two's Complement

Binary	Dec
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Number	2's Complement	
+1	0001	
-1	1111	-8 + 7
+5	0101	
-5	1011	-8 + 3
+0	0000	
-0	none	

What is the range that a 4-bit 2's-complement number can represent?

-2^3 to 2^3-1 , which is -8 to +7

-2^{k-1} to $2^{k-1}-1$ in general for k-bits

Sign-Extension in 2's Complement

- To make a k -bit number into a $k+1$ -bit number :
 - replicate sign bit (replace -2^k with $-2^{k+1} + 2^k$)
 - The new MSB is the new sign bit.

$$110_{2C} = \quad 1 \times (-2^2) + 1 \times 2^1 = -2_{10} \quad \begin{matrix} 3\text{-bit 2C} \\ \downarrow \end{matrix}$$

$$1110_{2C} = \quad 1 \times (-2^3) + 1 \times 2^2 + 1 \times 2^1 = -2_{10} \quad \begin{matrix} 4\text{-bit 2C} \\ \downarrow \end{matrix}$$

$$11110_{2C} = 1 \times (-2^4) + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 = -2_{10} \quad \begin{matrix} 5\text{-bit 2C} \\ \downarrow \end{matrix}$$

Sign-Extension for Positive Numbers

$$010_{2C} = 1 \times 2^1 = 2_{10}$$

$$0010_{2C} = 1 \times 2^1 = 2_{10}$$

$$00010_{2C} = 1 \times 2^1 = 2_{10}$$

Works for both positive and negative numbers

Negating a 2's Complement Number

1. Invert all the bits

$$X + \overline{X} = -1$$

2. Add 1

$$-X = \overline{X} + 1$$

Binary	Dec
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

$$+2_{10} = 0010_{2C} \rightarrow 1101 + 1 = 1110_{2C} = -2_{10}$$

$$-2_{10} = 1110_{2C} \rightarrow 0001 + 1 = 0010_{2C} = +2_{10}$$

Two's Complement Addition

$$\begin{array}{r} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\ 1011 \\ + 1111 \\ \hline 1010 \end{array} \quad \begin{array}{l} (-5_{10}) \\ (-1_{10}) \\ (-6_{10}) \end{array}$$

Addition operation is same as for unsigned.
Same rules, same procedure.
This is the biggest advantage
of two's complement numbers.

$$\begin{array}{r} \overset{0}{0} \overset{0}{1} \\ 0101 \\ + 0001 \\ \hline 0110 \end{array} \quad \begin{array}{l} (+5_{10}) \\ (+1_{10}) \\ (+6_{10}) \end{array}$$

For two's complement addition/subtraction,
you always ignore the carry out of the MSB

$$\begin{array}{r} \overset{1}{1} \overset{1}{0} \\ 0110 \\ + 1111 \\ \hline 0101 \end{array} \quad \begin{array}{l} (+6_{10}) \\ (-1_{10}) \\ (+5_{10}) \end{array}$$

Interpretation of operands and
results are different

Two's Complement Overflow

- All representations can overflow
 - Focus on 2's complement here

$$\begin{array}{r} \text{1 0 0 0} \\ & 1011 & (-5_{10}) \\ + & 1100 & (-4_{10}) \\ \hline & 0111 & (+7_{10}) \quad ?? \end{array}$$

Binary	Dec
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

The correct answer (-9) cannot be represented by 4 bits
The correct answer is 10111

2's Complement Overflow

- Can you use the leftmost carry-out as a new MSB?

$$\begin{array}{r} \text{1 0 0 0} \\ \text{1011} \quad (-5_{10}) \\ + \text{1100} \quad (-4_{10}) \\ \hline \text{10111} \quad (-9_{10}) \end{array}$$

$$\begin{array}{r} \text{0 0 0 0} \\ \text{1000} \quad (-8_{10}) \\ + \text{0111} \quad (+7_{10}) \\ \hline \text{01111} \quad (+15_{10}) \quad ?? \end{array}$$

Works here...

but does NOT work here...

- The answer is no
- In 2's complement, the carry must be discarded.

Handling Overflow

1. Sign-extend the operands
2. Re-do the addition

$$\begin{array}{rcl} & \begin{smallmatrix} 0 & 0 & 0 & 0 \end{smallmatrix} & \\ \begin{smallmatrix} 1 & 1 & 0 & 0 \end{smallmatrix} & (-8_{10}) & \\ + \begin{smallmatrix} 0 & 0 & 1 & 1 \end{smallmatrix} & (+7_{10}) & \\ \hline \begin{smallmatrix} 1 & 1 & 1 & 1 \end{smallmatrix} & (-1_{10}) & \end{array}$$

This always works

When Can Overflow Occur?

- Adding two positive numbers
 - yes
- Adding two negative numbers
 - yes
- Adding a positive to a negative
 - no
- Adding a negative to a positive
 - no

General Handling of Overflow

- Adding two k -bit numbers gives $k+1$ bit result
- Two options for handling:
 1. Sign-extend both k -bit 2's complement numbers to $k+1$ bits *before* adding to prevent possible overflow.
 2. Add and keep only k bits (throw out extra bit)
 - i. Detect overflow and signal an error
 - ii. Ignore the overflow
- The choice is up to the designer

Detecting Overflow

- 2's Complement overflow occurs if:
 - Adding two positive numbers gives a negative result
 - Adding two negative numbers gives a positive result
- 2's Complement overflow will never occur when adding a positive and a negative number together

Lecture 18

Arithmetic Circuits
Half-Adders, Full-Adders
Carry Ripple Adders
Subtractors

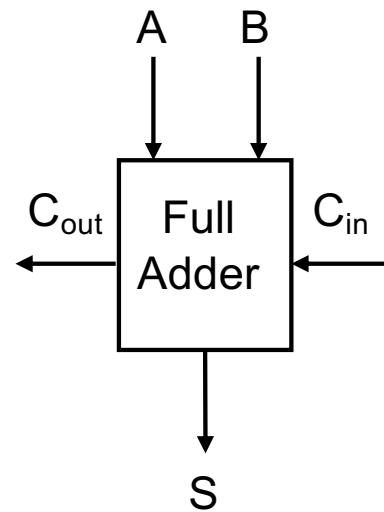
Binary Adder

$$\begin{array}{r} 0 \ 1 \ 1 \ 0 \\ 10010 \\ +00111 \\ \hline 11001 \end{array}$$



Inputs: Operand A
Operand B
Carry In

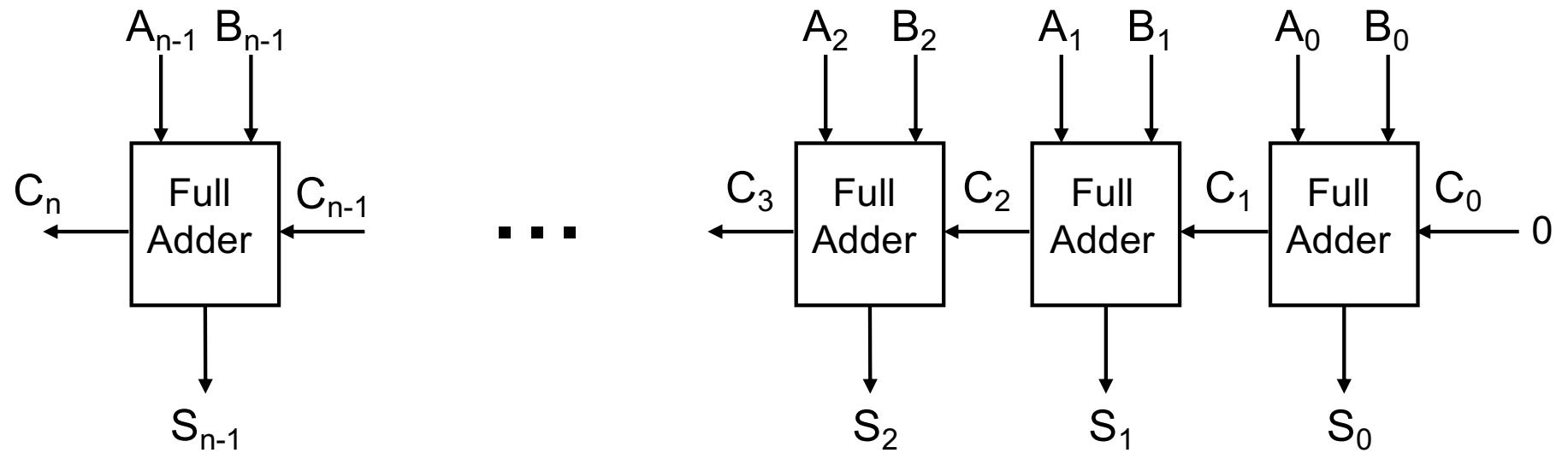
Outputs: Sum
Carry Out



The Full Adder is the basis of all computer arithmetic circuits.

Binary Adder

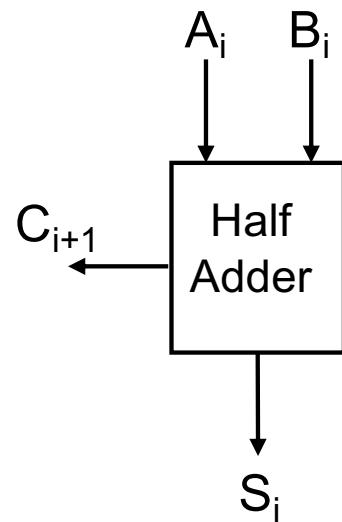
- An example of an iterative network



This type of adder is called a **ripple-carry adder** because the carry ripples through from cell to cell

Half Adder Derivation

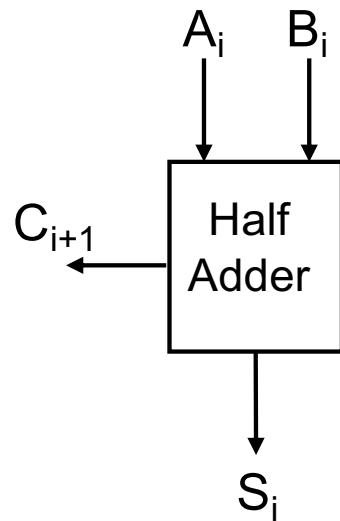
$$2C_{i+1} + S_i = A_i + B_i \text{ (ADD)}$$



A_i	B_i	C_{i+1}	S_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder Derivation

$$2C_{i+1} + S_i = A_i + B_i \text{ (ADD)}$$

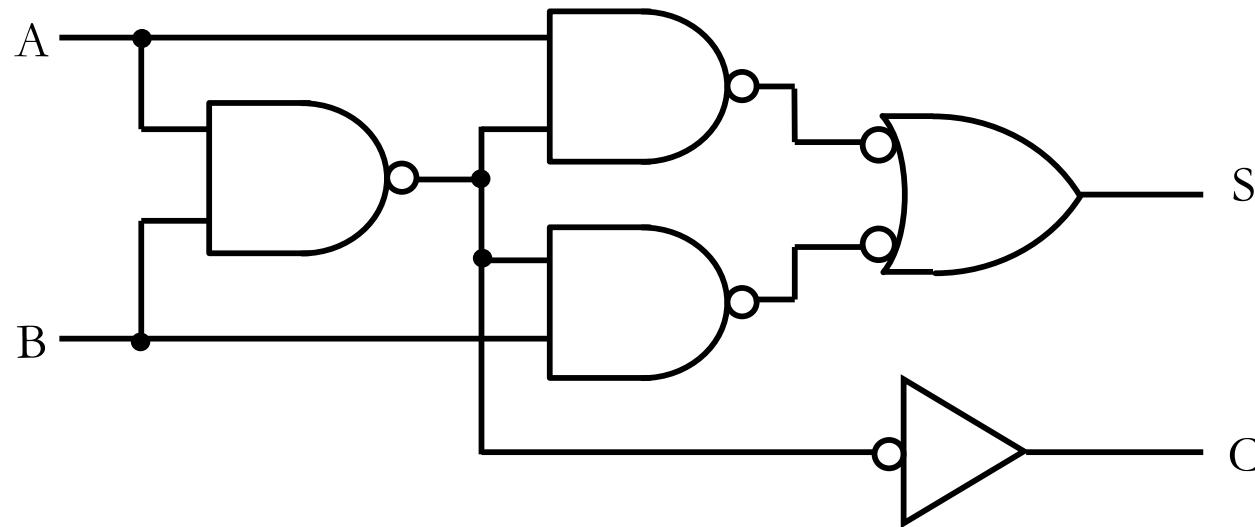
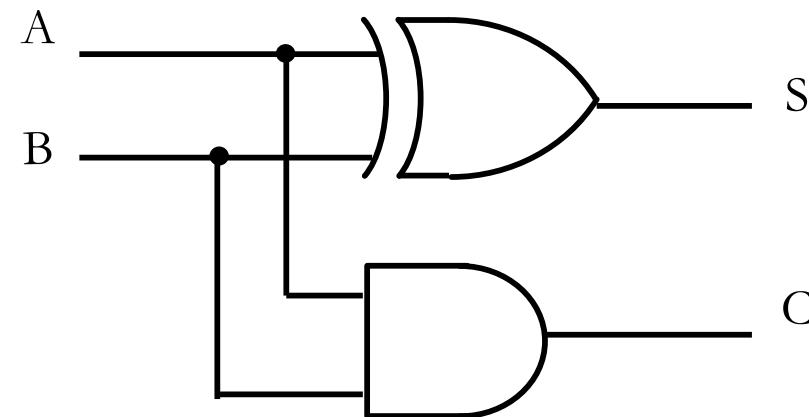


A_i	B_i	C_{i+1}	S_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C = AB$$

Half Adder

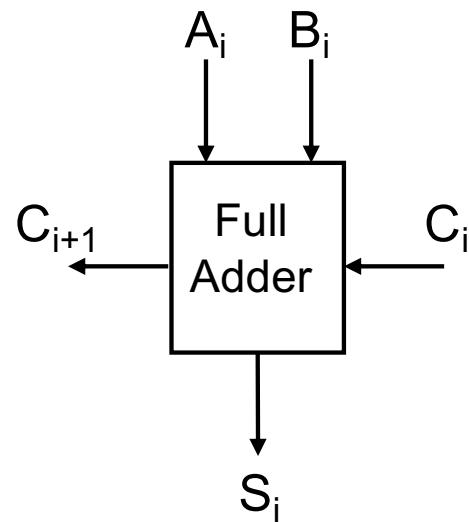
$$S = A \oplus B$$
$$C = AB$$



All
NAND
Gates

Full Adder Derivation

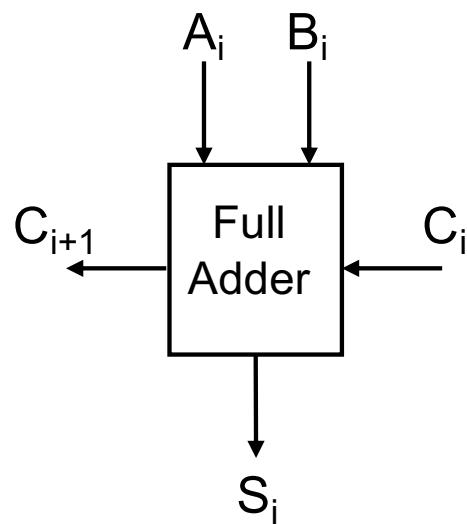
$$2C_{i+1} + S_i = A_i + B_i + C_i \text{ (ADD)}$$



A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Full Adder Derivation

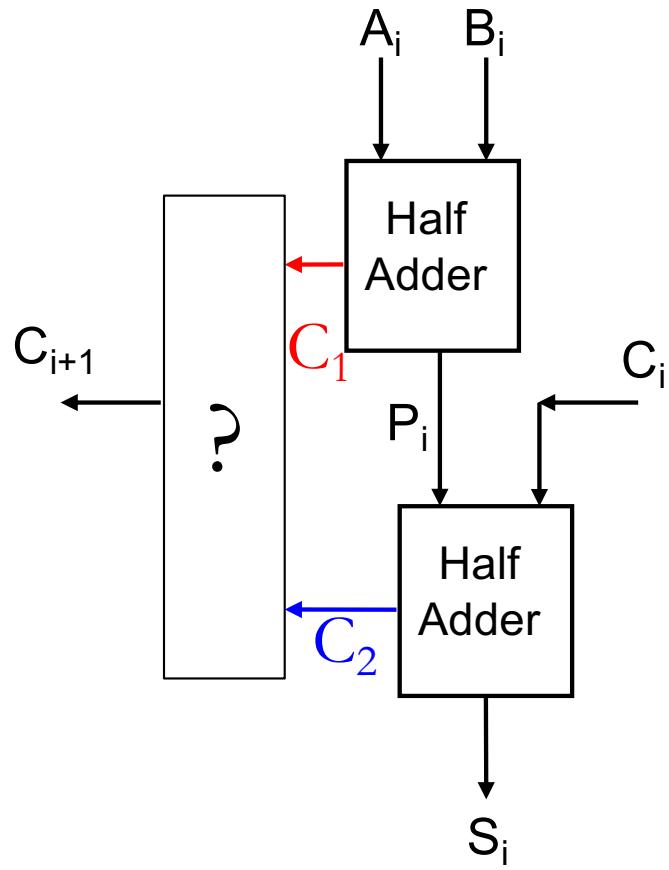
$$2C_{i+1} + S_i = A_i + B_i + C_i \text{ (ADD)}$$



A_i	B_i	C_i	C_{i+1}	S_i
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

Full Adder Derivation

$$2C_{i+1} + S_i = ((A_i + B_i) + C_i) \quad (\text{ADD})$$

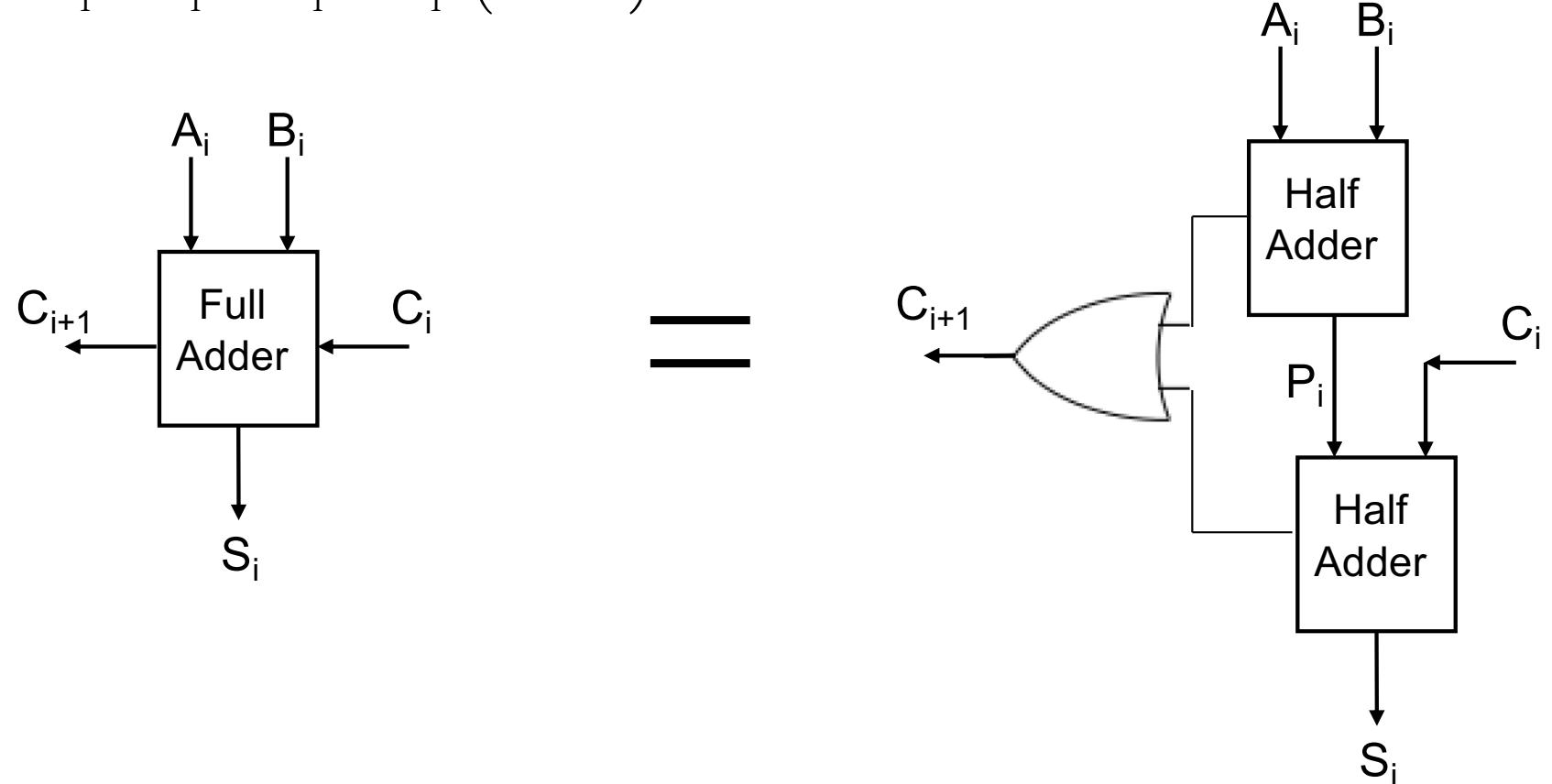


A_i	B_i	C_i	P_i	C_1	C_2	C_{i+1}	S_i
0	0	0				0	0
0	0	1				0	1
0	1	0				0	1
0	1	1	1			1	0
1	0	0				0	1
1	0	1	1			1	0
1	1	0		0		1	0
1	1	1		1		1	1

$$C_{i+1} = C_1 + C_2 \quad (\text{OR})$$

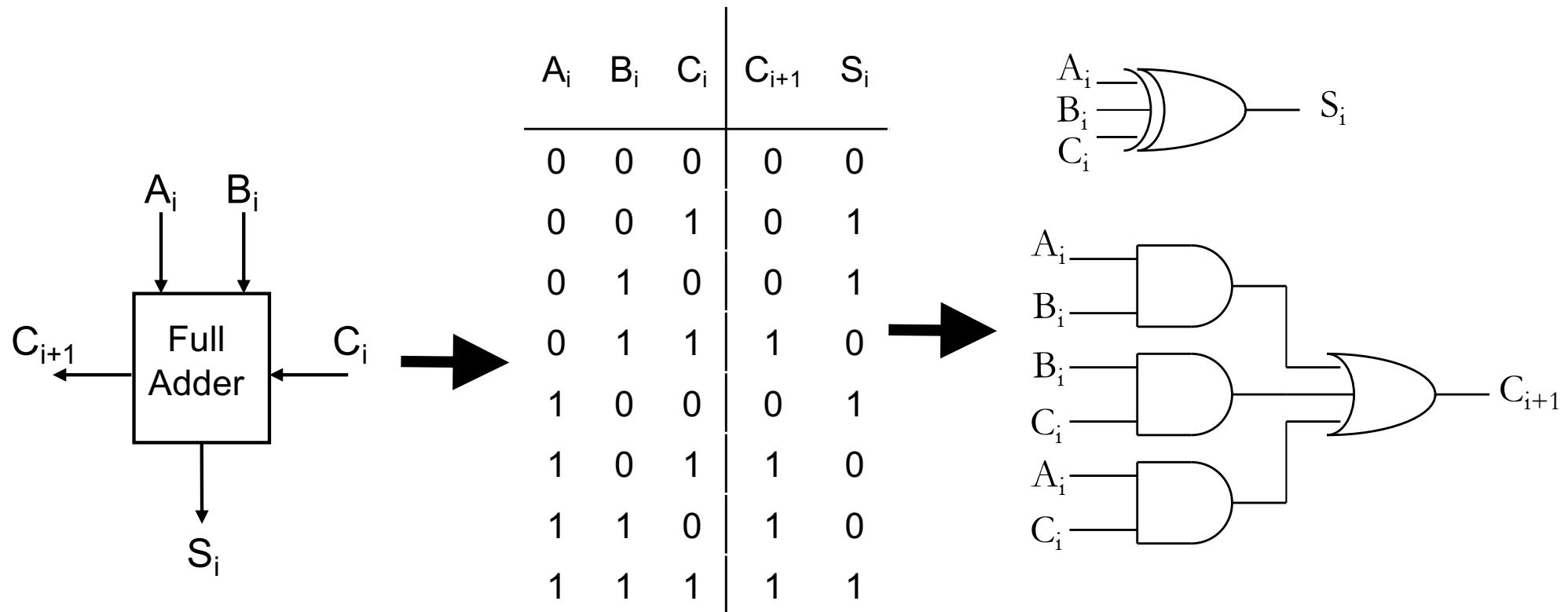
Full Adder Circuit

$$2C_{i+1} + S_i = A_i + B_i + C_i \text{ (ADD)}$$



Full Adder Circuit

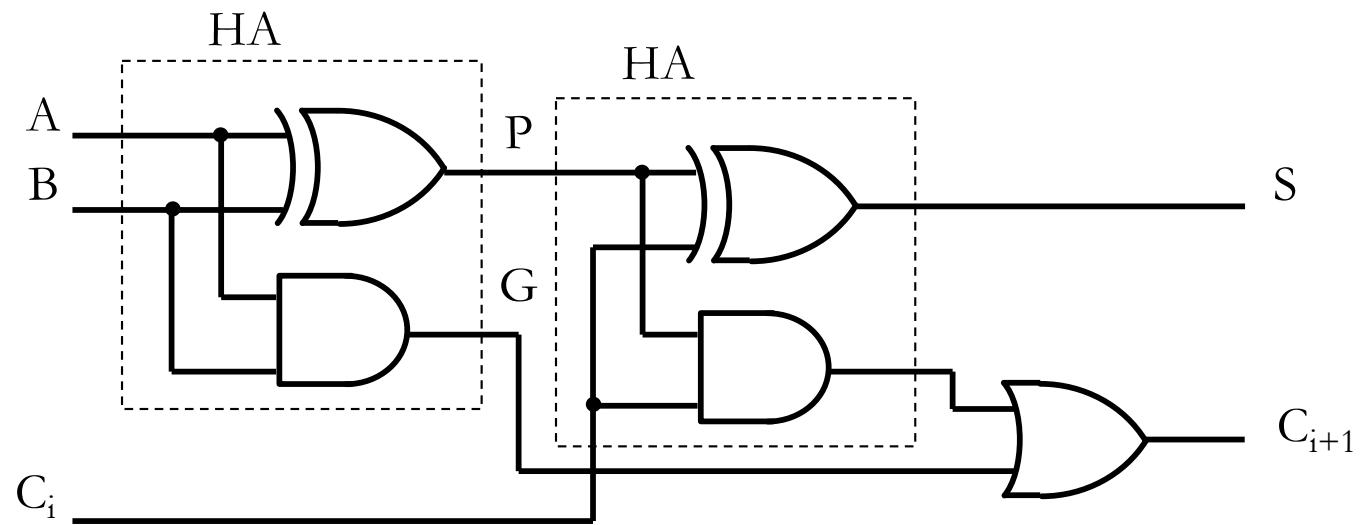
$$2C_{i+1} + S_i = A_i + B_i + C_i \text{ (ADD)}$$



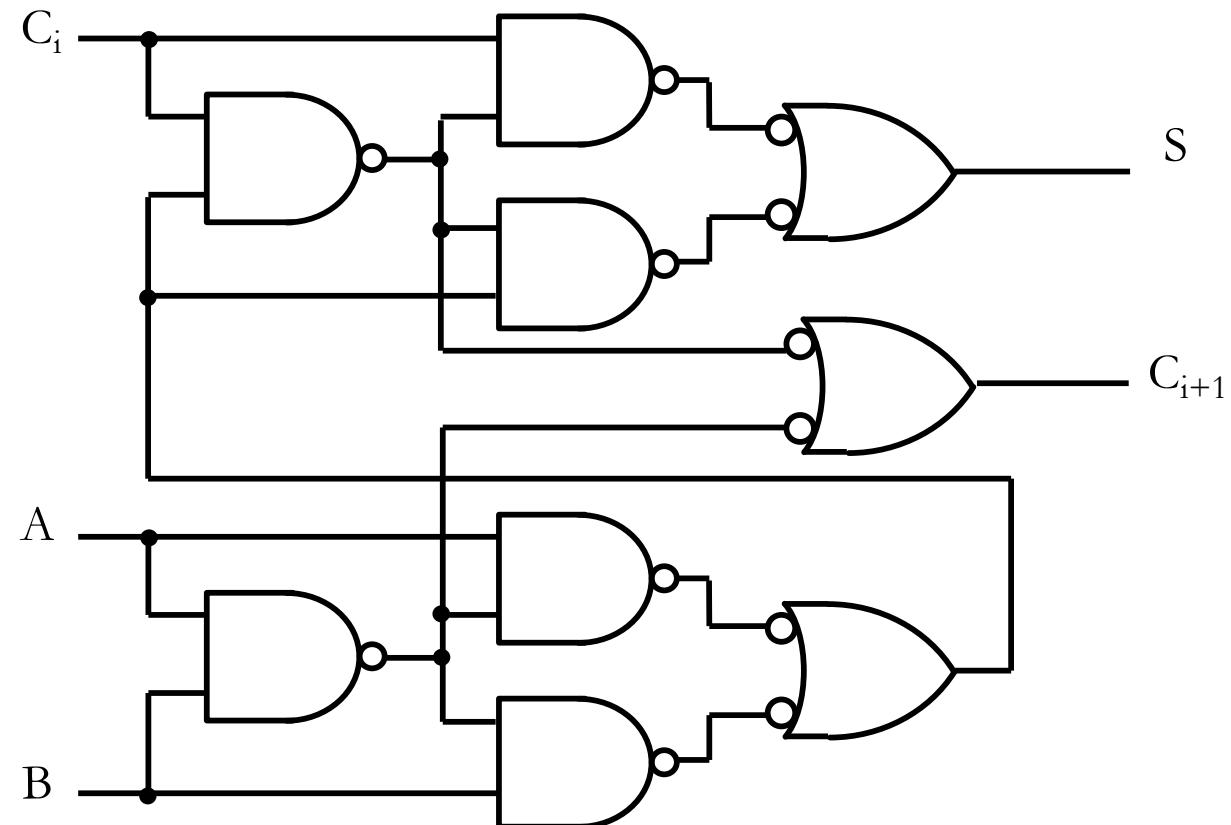
$$S = AB' C_i' + A' B' C_i + ABC + A' BC_i'$$

$$C_{i+1} = AC_i + AB + BC_i$$

Full Adder Circuit

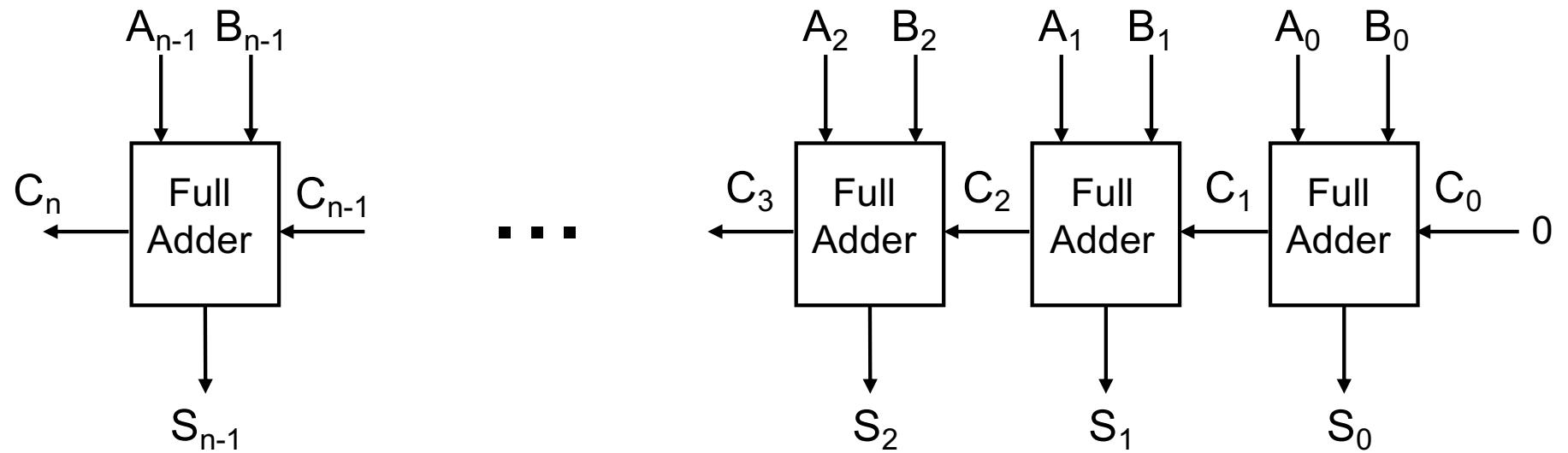


Full Adder Circuit



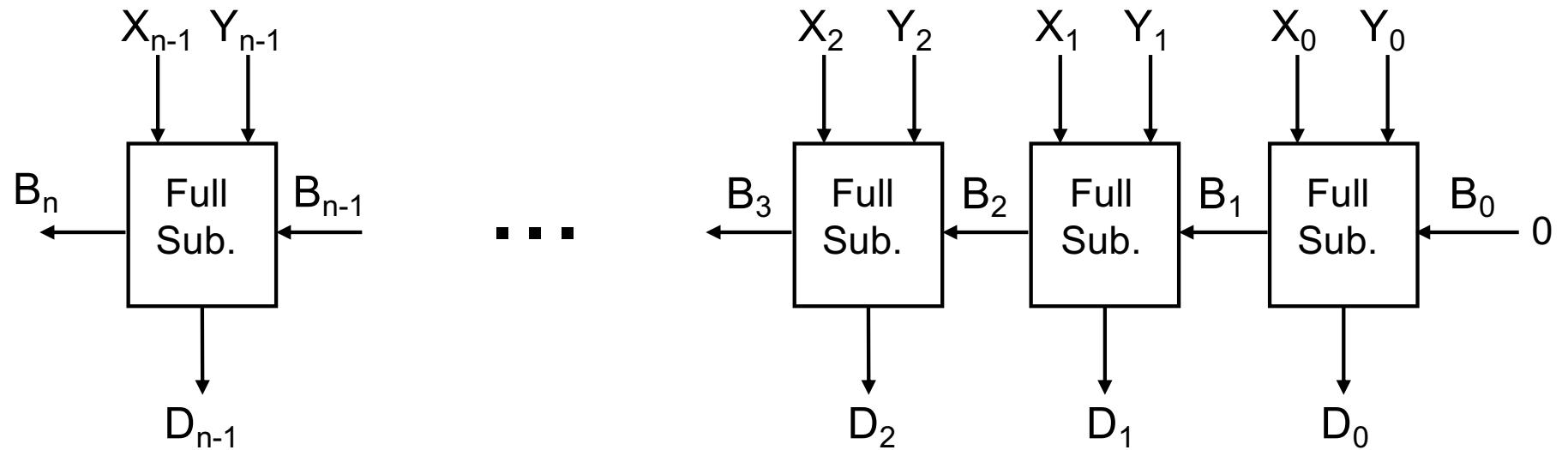
All NAND
gate circuit

Ripple-Carry Binary Adder



But what about subtraction?

Ripple-Borrow Binary Subtractor



D is the difference ($X - Y$)

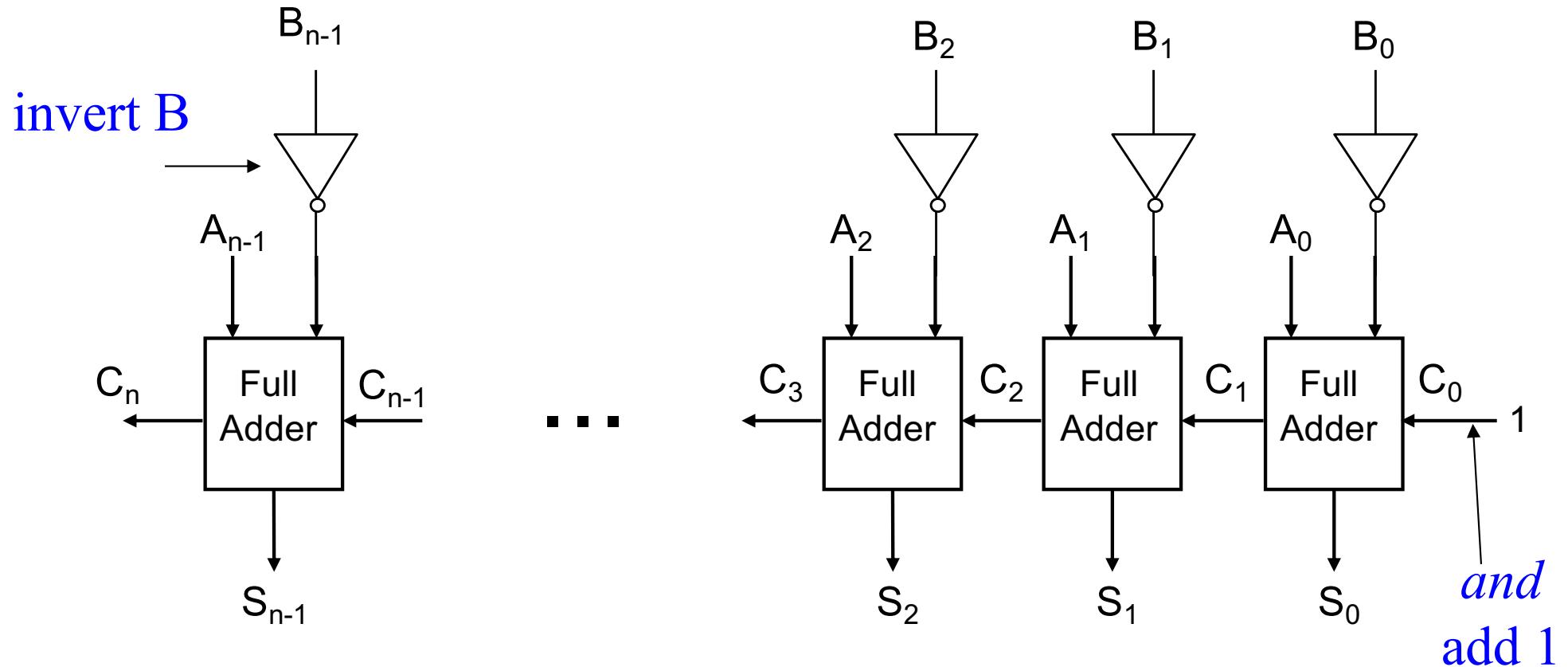
B is the borrow signal

Equations generated
similarly to full adder

Binary Subtractor Using Adder

- Good News! – We don't have to design a new circuit to do subtraction.
- Recall that $A - B = A + (-B)$
- A subtractor can be easily implemented by negating B and adding it to A
- Negate B by inverting the B bits and adding 1
$$-B = \sim B + 1 \quad A - B = A + (-B) = A + \sim B + 1$$
- Adders can add *and* subtract.
This is a big advantage of 2's-complement arithmetic.

Ripple-Carry Binary Subtractor (using Full Adders)



$$S = A - B = A + (-B) = A + \sim B + 1$$

$-B$ is generated by inverting and adding 1

Lecture 19

The 555 Timer

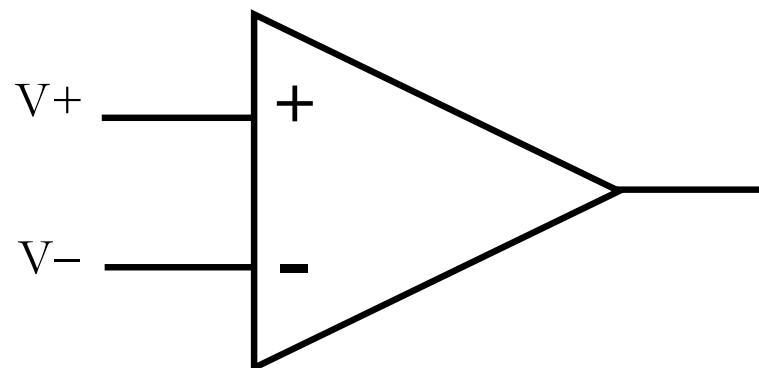
What is a 555 Timer?

- The 555 timer is an 8-pin IC that is capable of producing accurate time delays and/or oscillators.
- Introduced in 1971 by Signetics, the 555 is still in widespread use due to its ease of use, low price, and stability.
- In the **time delay mode**, the **delay is controlled by one external resistor and capacitor.**
- In the **oscillator mode**, the **frequency of oscillation and duty cycle are both controlled with two external resistors and one capacitor.**



Op Amp Comparator

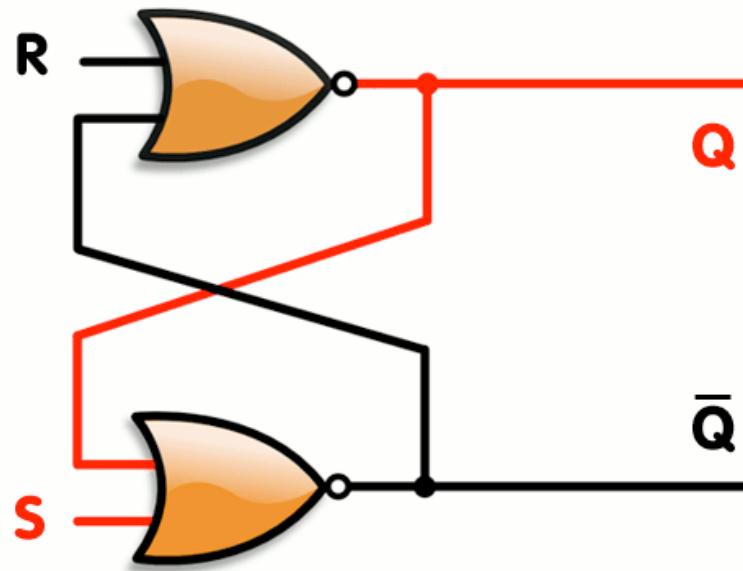
- Open loop (no feedback) means the Op Amp operates in saturation regions.
(Output = V_{cc+} or V_{cc-})
- The comparator compares 2 analog voltages and provides a digital output.



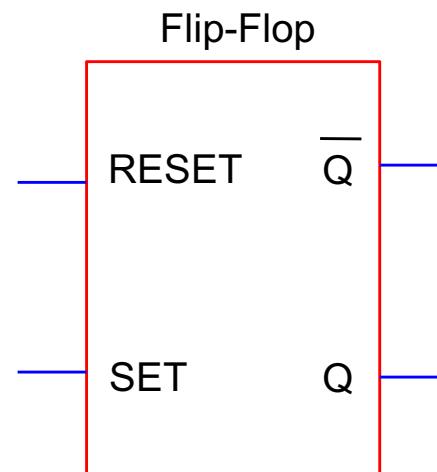
If $V_+ > V_-$, the output is a digital 1

If $V_- > V_+$, the output is a digital 0

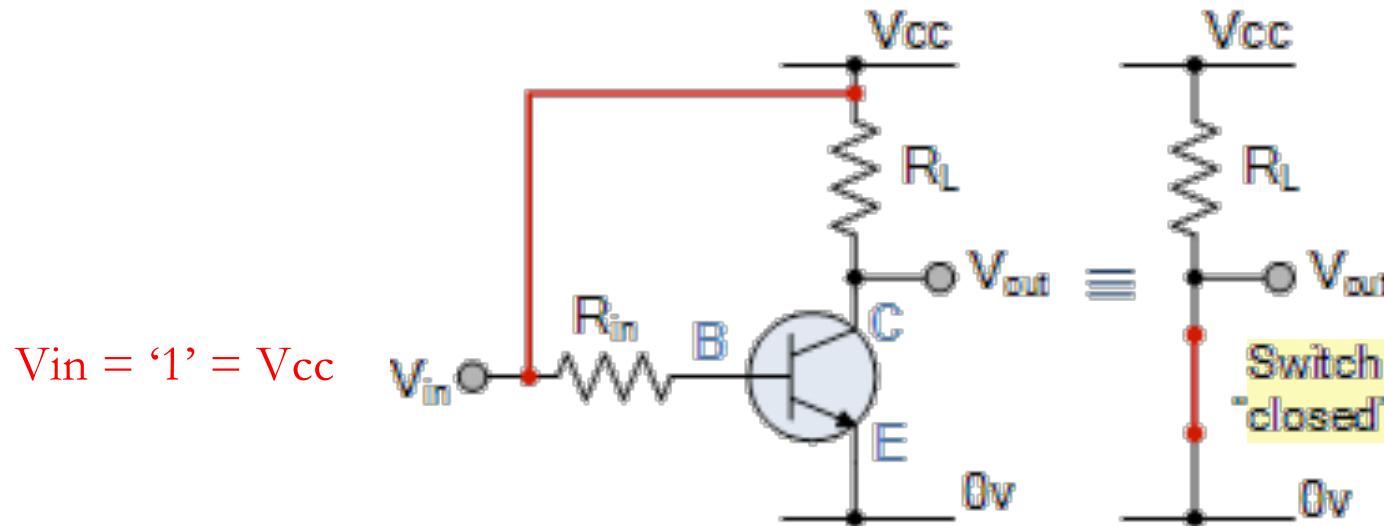
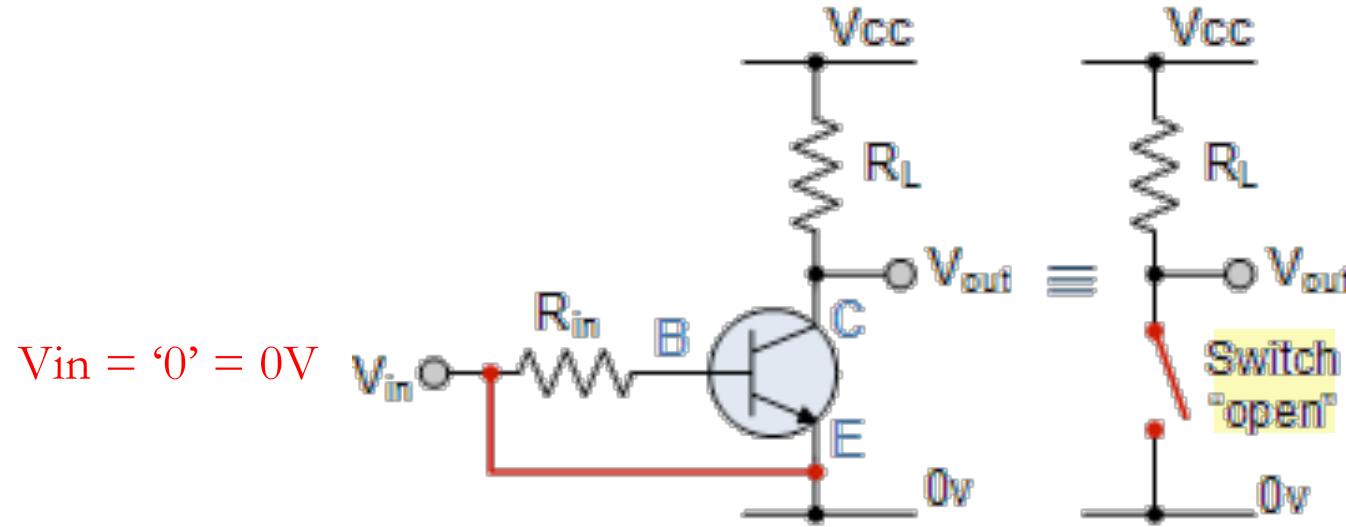
Set-Reset Flip-Flop



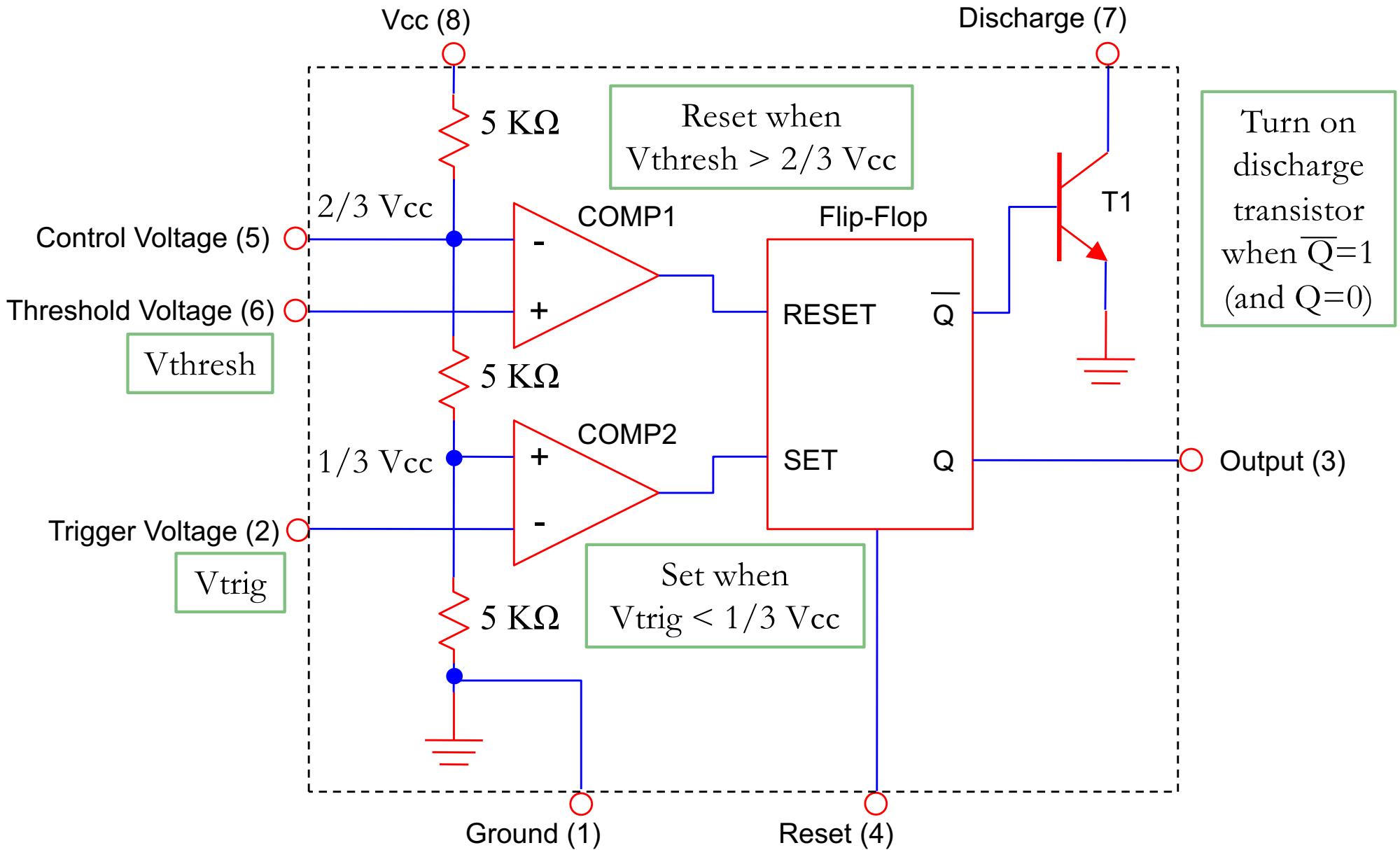
S	R	Q_{next}	Action
0	0	Q	hold state
0	1	0	reset
1	0	1	set
1	1	X	not allowed



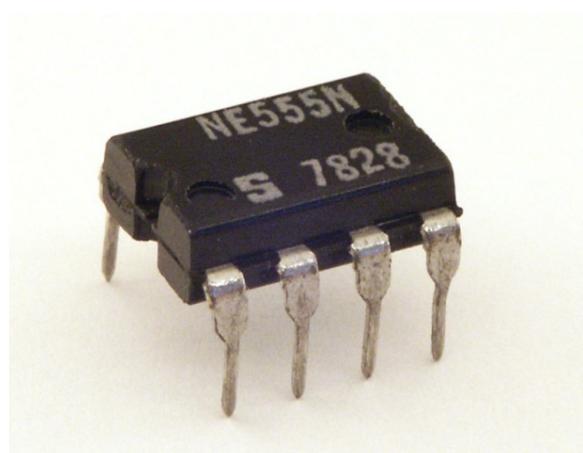
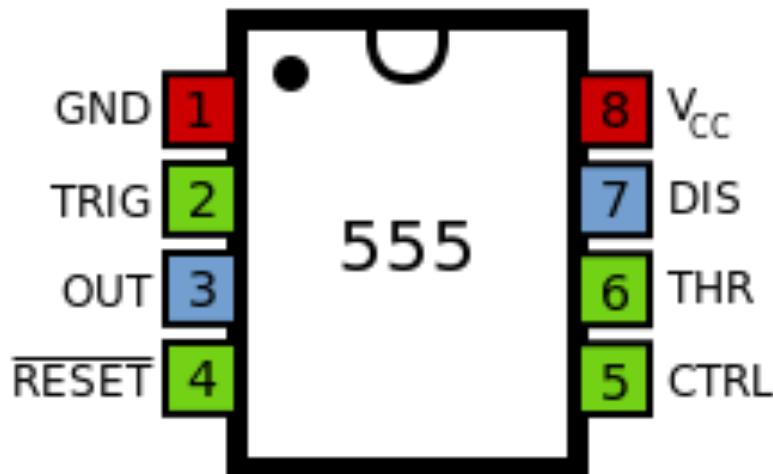
Transistor as an Electronic Switch



Block Diagram for a 555 Timer



555 Pinout



FAIRCHILD
SEMICONDUCTOR®

www.fairchildsemi.com

LM555/NE555/SA555 Single Timer

Features

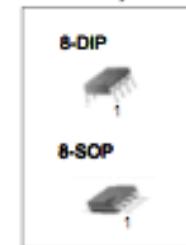
- High Current Drive Capability (200mA)
- Adjustable Duty Cycle
- Temperature Stability of 0.005%/°C
- Timing From μ Sec to Hours
- Turn off Time Less Than 2 μ Sec

Applications

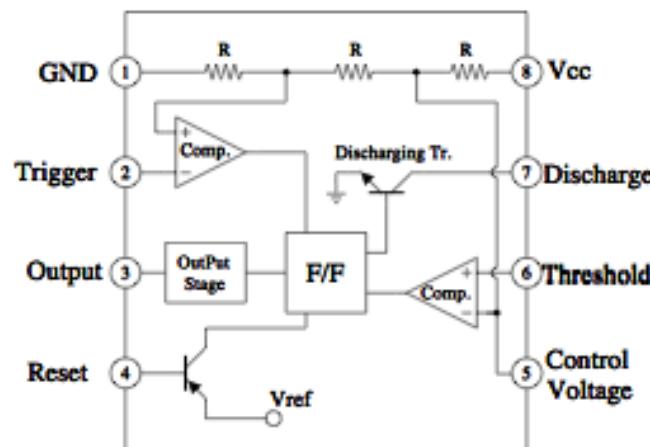
- Precision Timing
- Pulse Generation
- Time Delay Generation
- Sequential Timing

Description

The LM555/NE555/SA555 is a highly stable controller capable of producing accurate timing pulses. With a monostable operation, the time delay is controlled by one external resistor and one capacitor. With an astable operation, the frequency and duty cycle are accurately controlled by two external resistors and one capacitor.

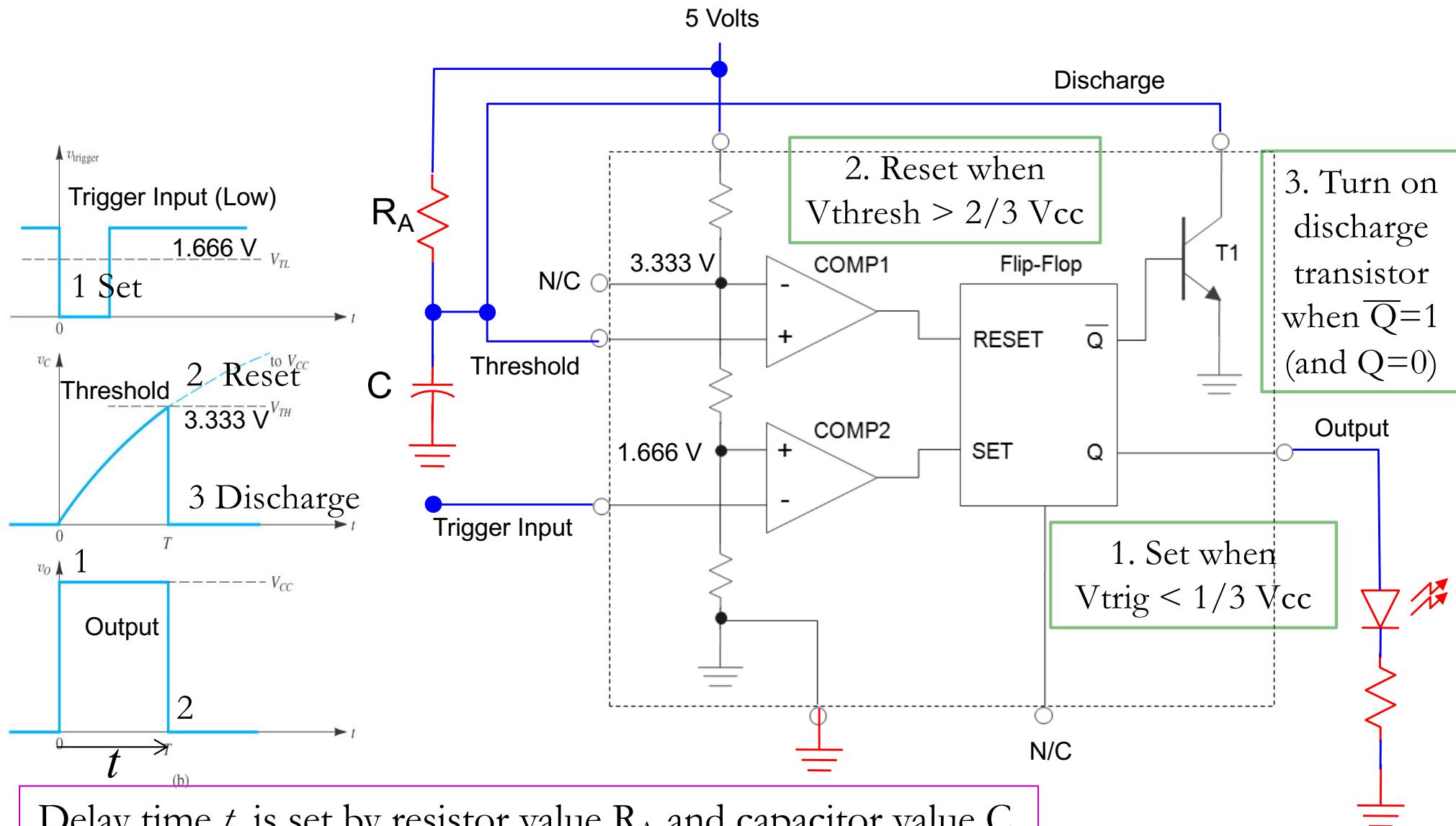


Internal Block Diagram



Schematic of a 555 Timer in Time Delay Mode

(Also called Monostable Mode)



Delay Time Design Equations

Calculations for the Delay Time

$$V_{Threshold}(t) = V_{Final} + (V_{Initial} - V_{Final}) e^{-\frac{t}{RC}}$$

$$\frac{2}{3}V_{CC} = V_{CC} + (0 - V_{CC}) e^{-\frac{t}{RC}}$$

$$-\frac{1}{3}V_{CC} = -V_{CC} e^{-\frac{t}{RC}}$$

$$\frac{1}{3} = e^{-\frac{t}{RC}}$$

$$\frac{1}{3} = e^{-\frac{t}{RC}}$$

$$\ln\left(\frac{1}{3}\right) = \ln\left(e^{-\frac{t}{RC}}\right)$$

$$-\ln(3) = -\frac{t}{RC}$$

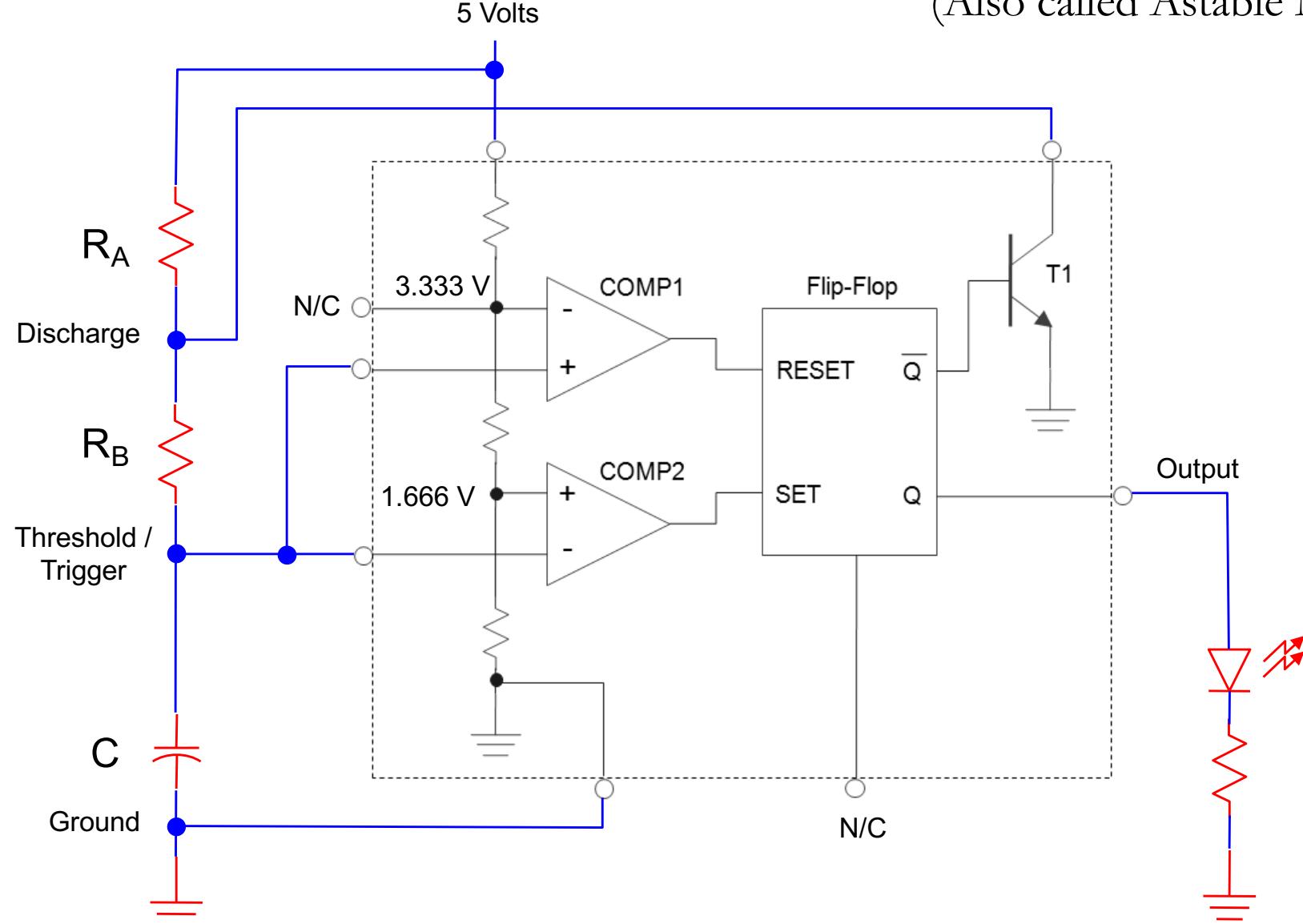
$$t = \ln(3) \cdot R_A C$$

$$t = 1.1 \cdot R_A C$$

$$(\ln(3) = 1.0986)$$

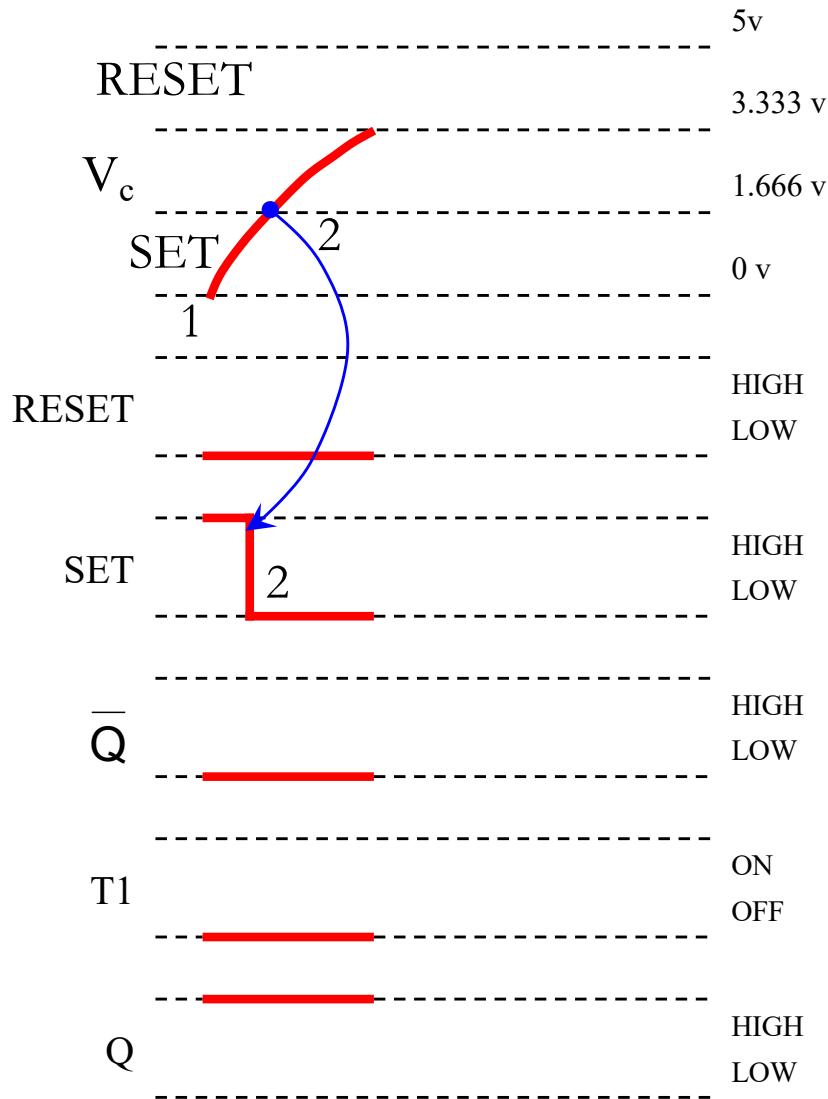
Schematic of a 555 Timer in Oscillator Mode

(Also called Astable Mode)

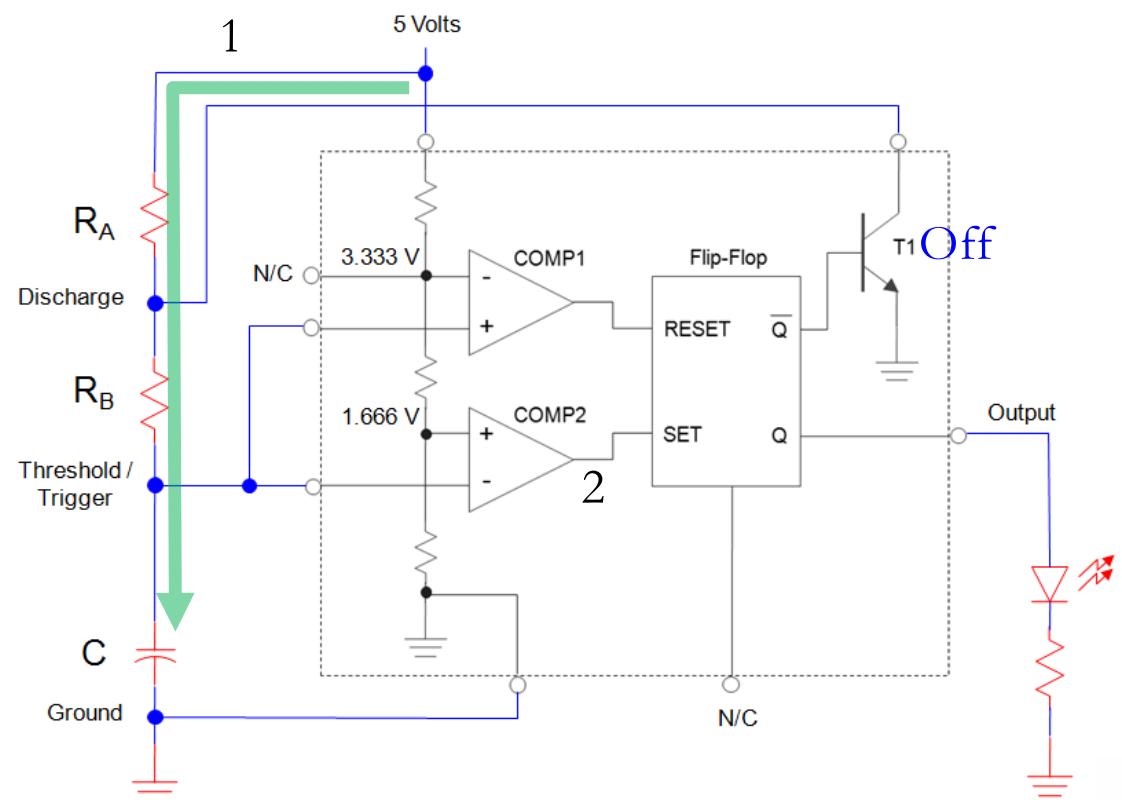


Detail Analysis of a 555 Oscillator

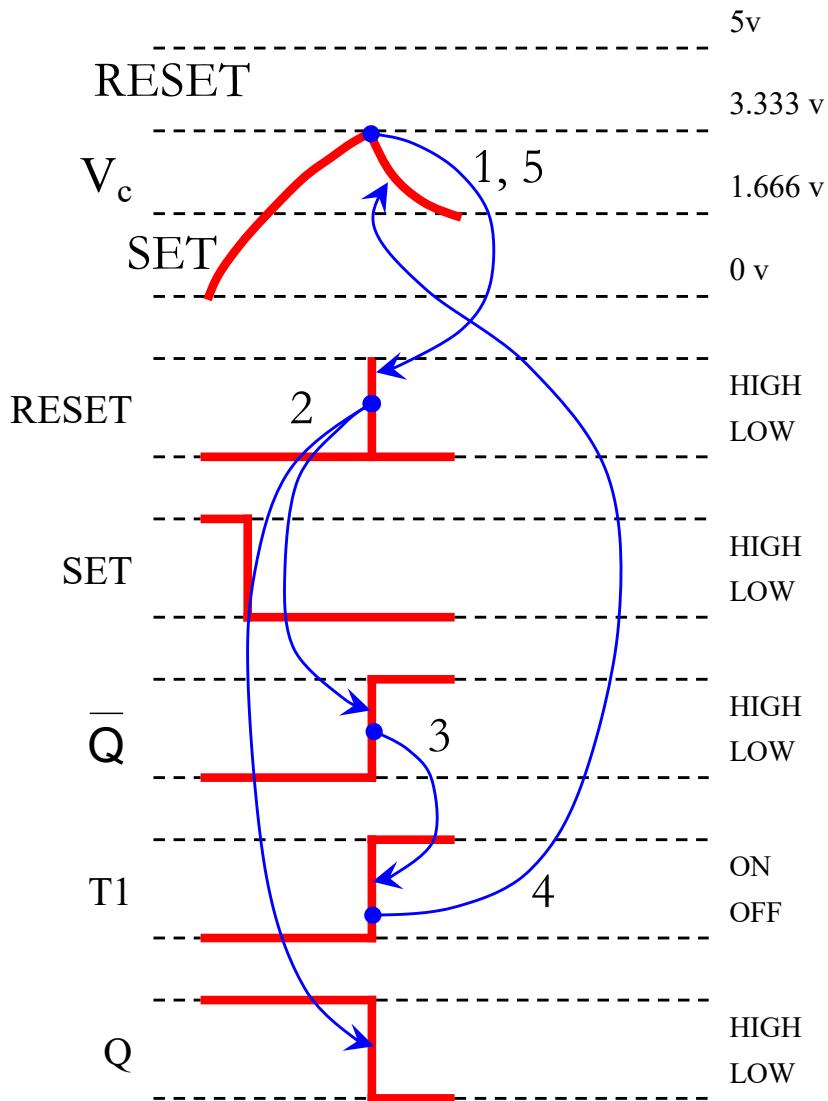
Start-up



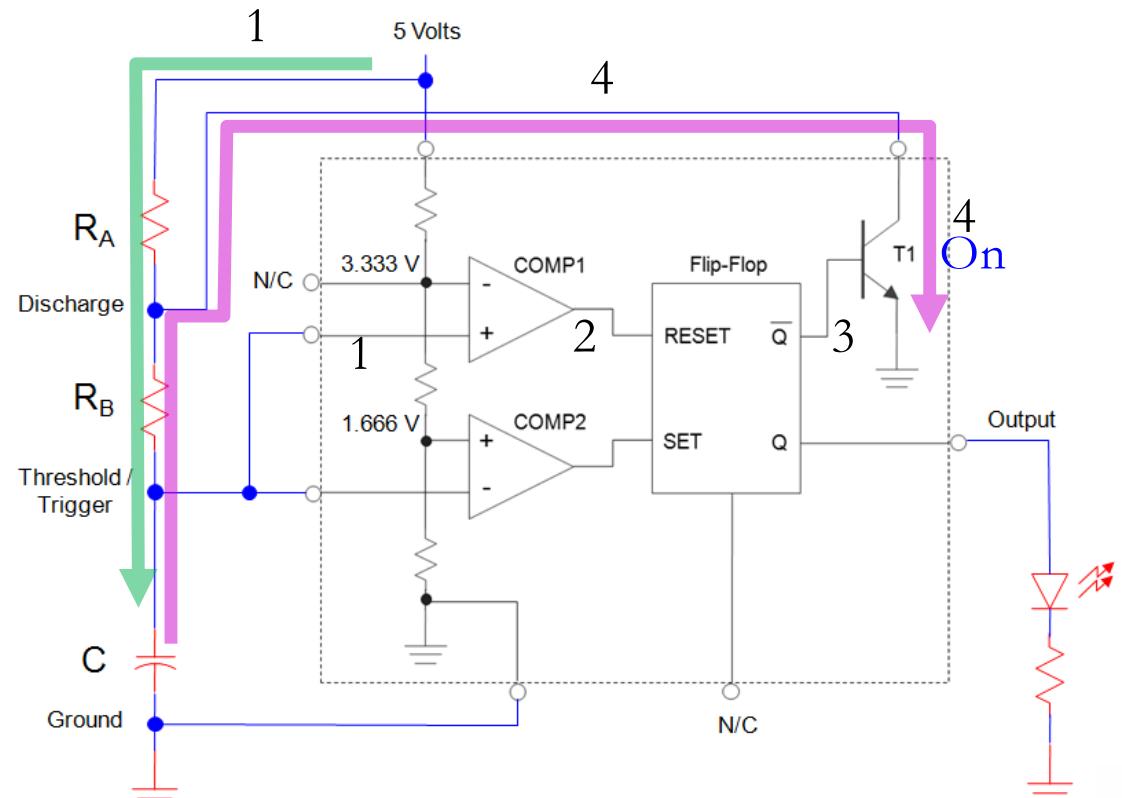
THE OUTPUT IS HIGH WHILE THE CAPACITOR IS CHARGING THROUGH $R_A + R_B$.



Detail Analysis of a 555 Oscillator

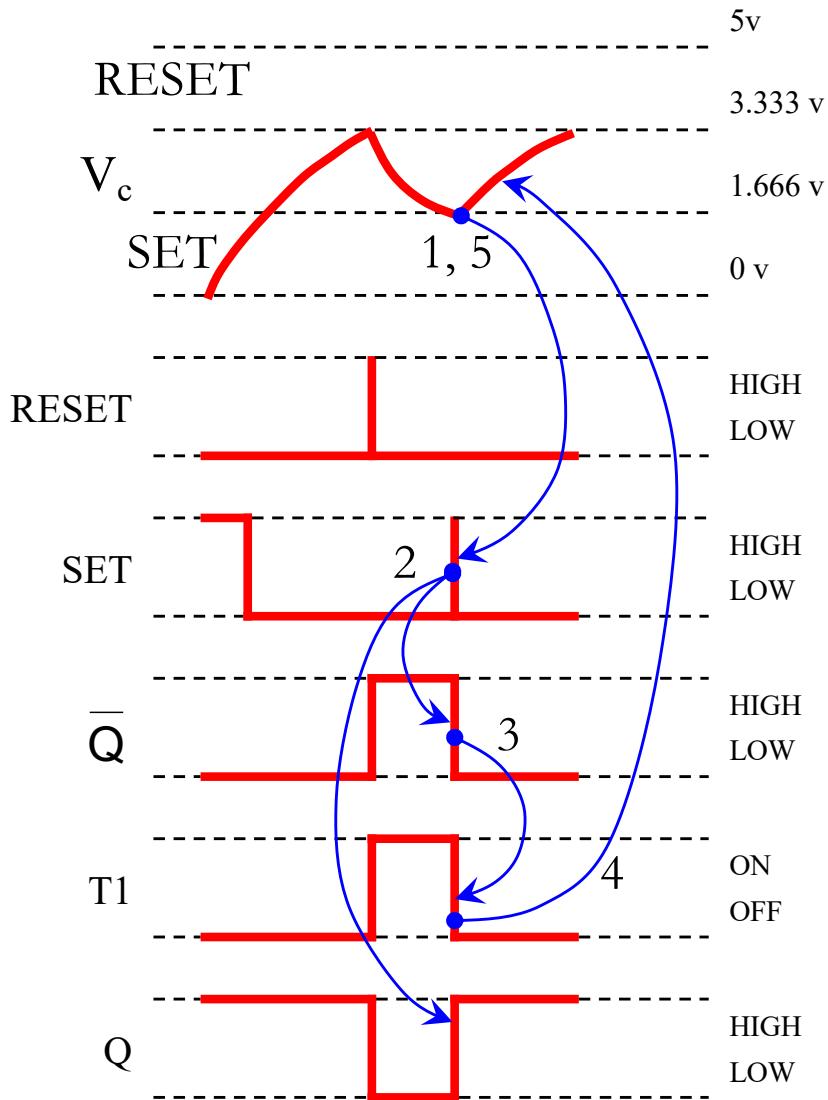


THE OUTPUT IS HIGH WHILE THE CAPACITOR IS CHARGING THROUGH $R_A + R_B$.

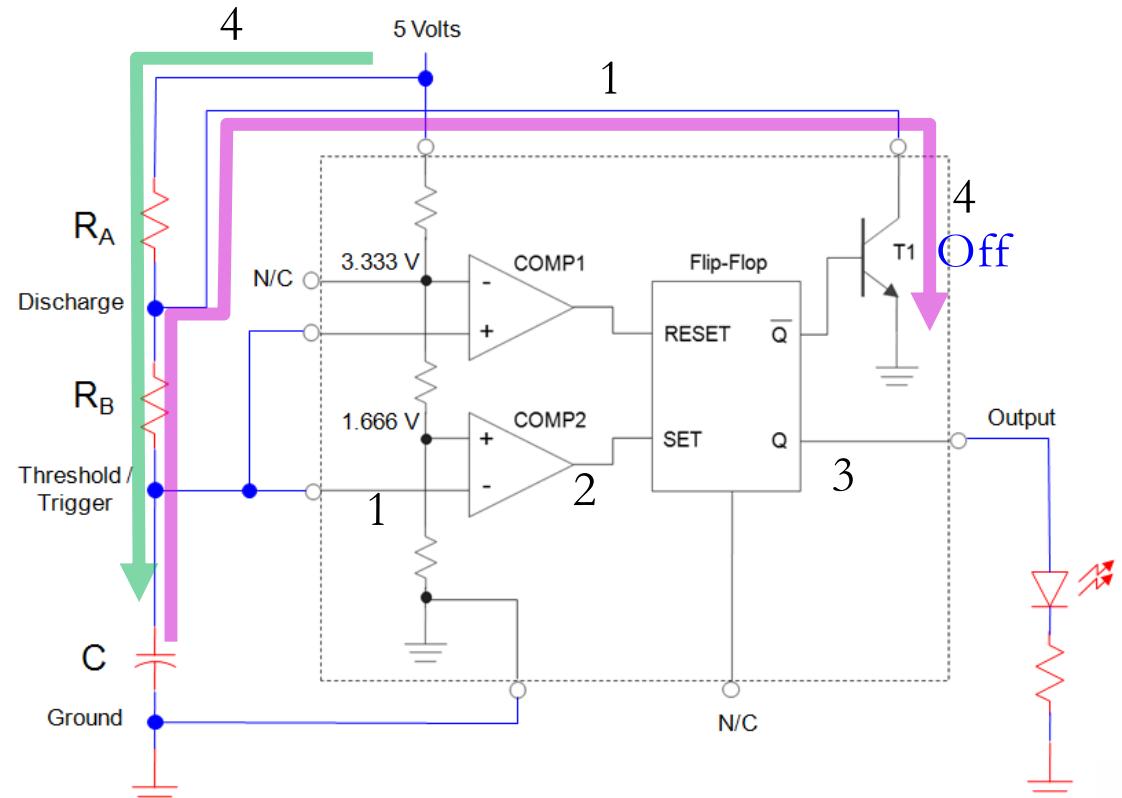


THE OUTPUT IS LOW WHILE THE CAPACITOR IS DISCHARGING THROUGH R_B .

Detail Analysis of a 555 Oscillator

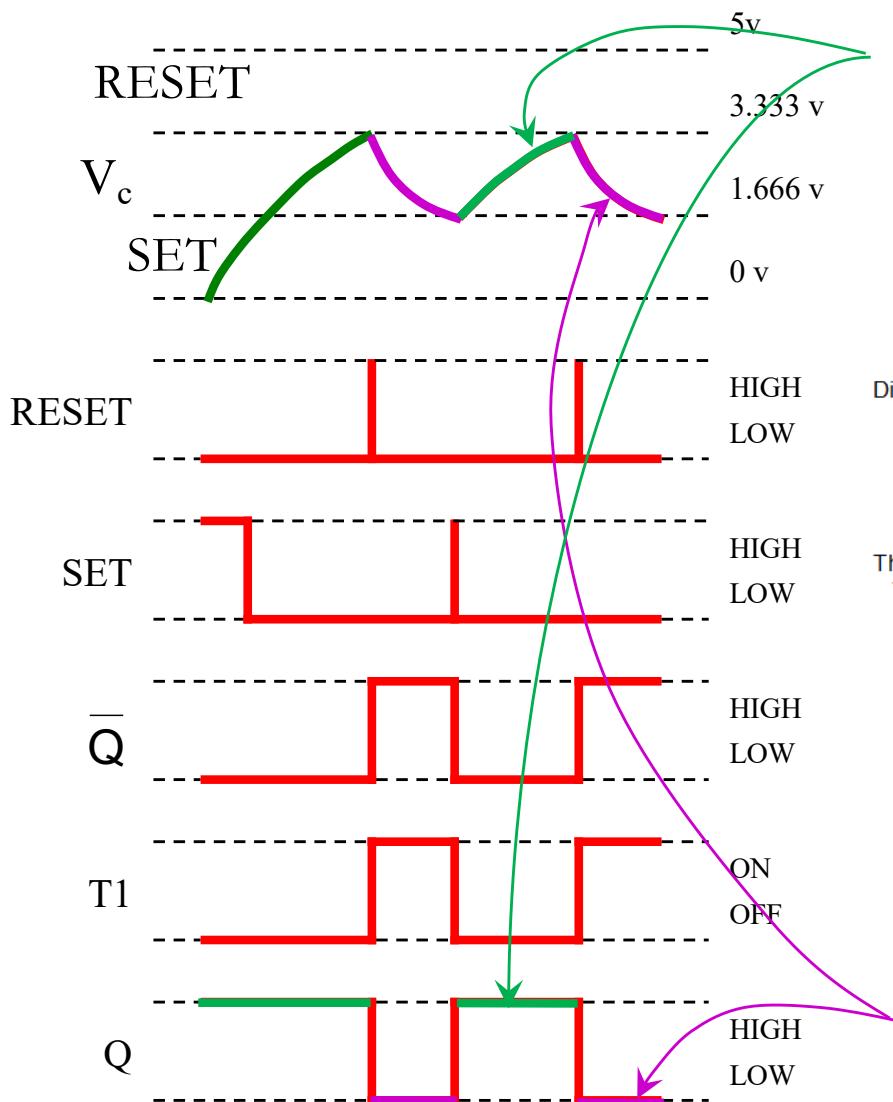


THE OUTPUT IS LOW WHILE THE CAPACITOR IS DISCHARGING THROUGH R_B .

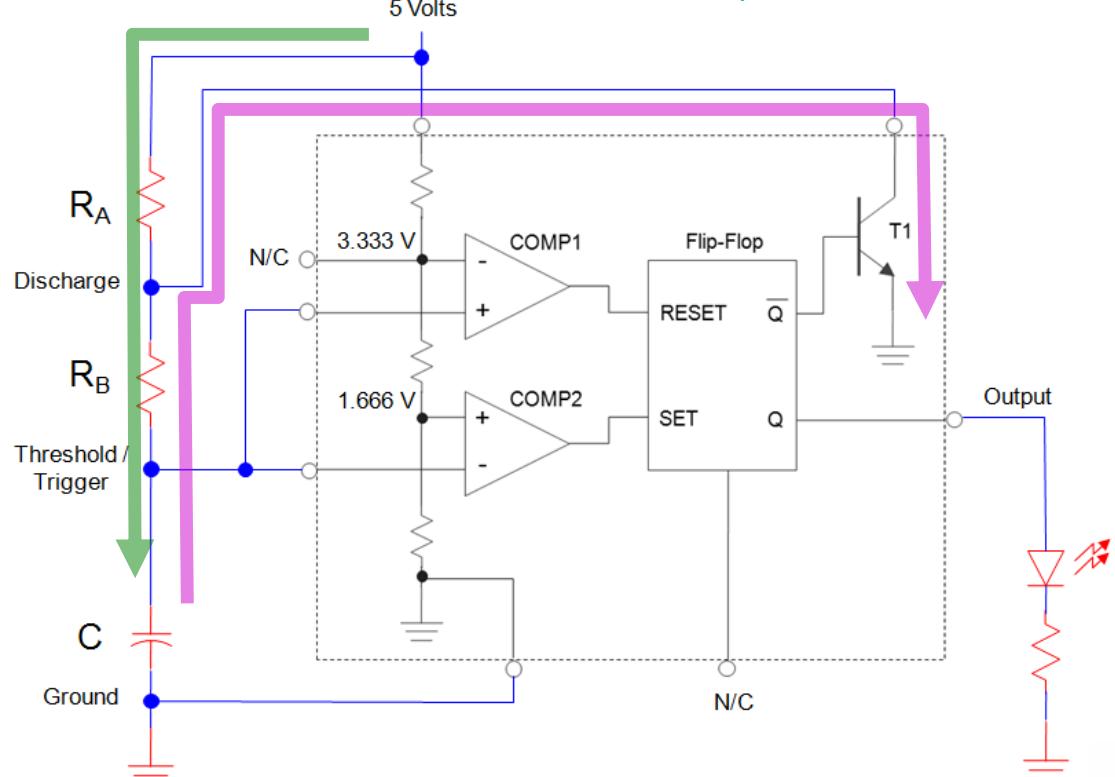


THE OUTPUT IS HIGH WHILE THE CAPACITOR IS CHARGING THROUGH $R_A + R_B$.

Detail Analysis of a 555 Oscillator



OUTPUT IS HIGH WHILE THE CAPACITOR IS CHARGING THROUGH $R_A + R_B$.



OUTPUT IS LOW WHILE THE CAPACITOR IS DISCHARGING THROUGH R_B .

555 Timer Design Equations

t_{HIGH} : Calculations for the Oscillator's HIGH Time

$$V_C(t) = V_{Final} + (V_{Initial} - V_{Final}) e^{-\frac{t}{RC}}$$

$$\frac{2}{3}V_{CC} = V_{CC} + \left(\frac{1}{3}V_{CC} - V_{CC}\right) e^{-\frac{t}{RC}}$$

$$\frac{2}{3}V_{CC} = V_{CC} - \frac{2}{3}V_{CC} e^{-\frac{t}{RC}}$$

$$-\frac{1}{3}V_{CC} = e^{-\frac{t}{RC}}$$

$$-\frac{2}{3}V_{CC}$$

$$\frac{1}{2} = e^{-\frac{t}{RC}}$$

$$\frac{1}{2} = e^{-\frac{t}{RC}}$$

$$\ln\left(\frac{1}{2}\right) = \ln\left(e^{-\frac{t}{RC}}\right)$$

$$-\ln(2) = -\frac{t}{RC}$$

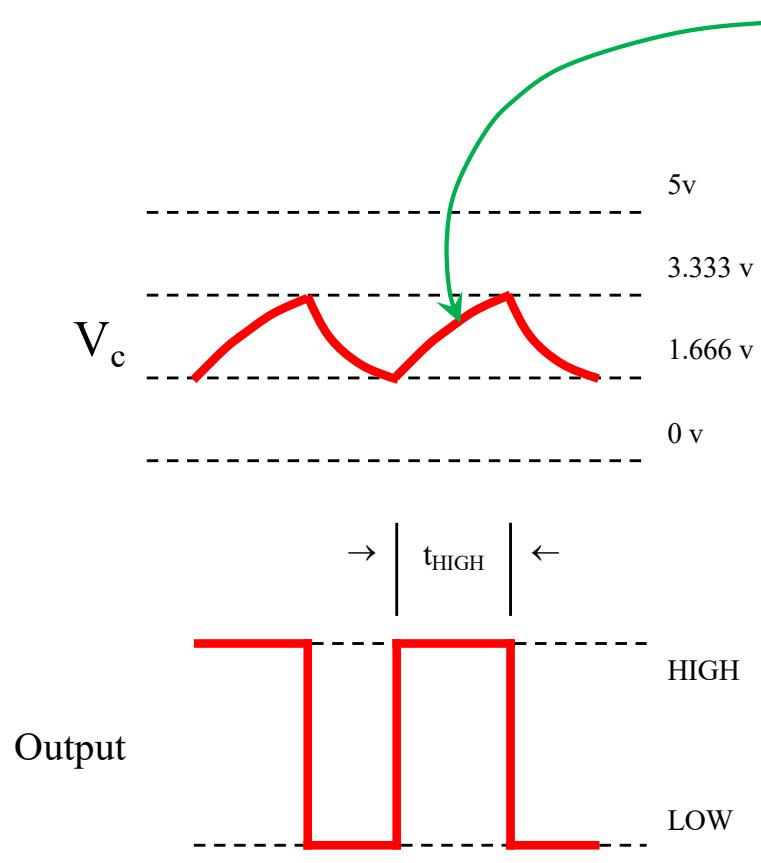
$$t_{HIGH} = \ln(2) \cdot RC$$

$$t_{HIGH} = 0.693 \cdot (R_A + R_B)C$$

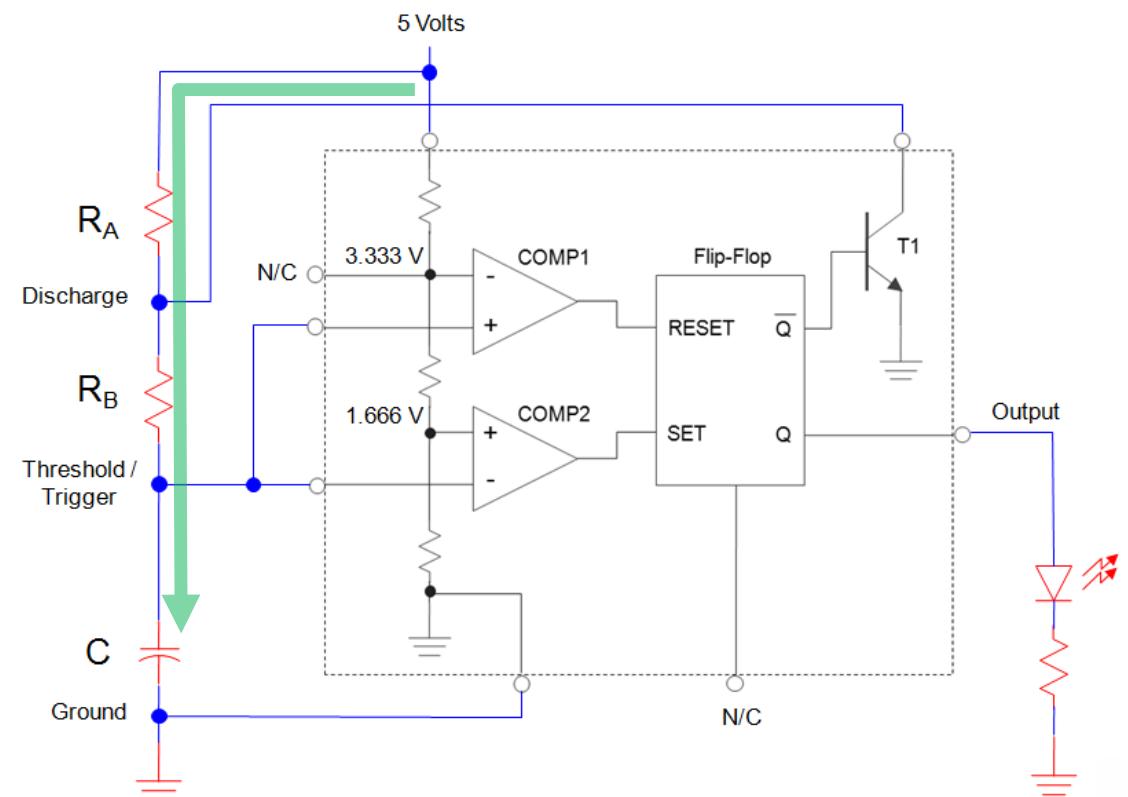
$$(\ln(2) = 0.693)$$

555 Timer Design Equations

t_{HIGH} : Calculations for the Oscillator's HIGH Time



THE OUTPUT IS HIGH WHILE THE CAPACITOR IS CHARGING THROUGH $R_A + R_B$.



$$t_{HIGH} = 0.693 \cdot (R_A + R_B)C$$

555 Timer Design Equations

t_{LOW} : Calculations for the Oscillator's LOW Time

$$V_C(t) = V_{Final} + (V_{Initial} - V_{Final}) e^{-\frac{t}{RC}}$$

$$\frac{1}{3}V_{CC} = 0 + \left(\frac{2}{3}V_{CC} - 0\right) e^{-\frac{t}{RC}}$$

$$\frac{1}{3}V_{CC} = \left(\frac{2}{3}V_{CC}\right) e^{-\frac{t}{RC}}$$

$$\frac{1}{3}V_{CC} = e^{-\frac{t}{RC}}$$

$$\frac{2}{3}V_{CC}$$

$$\frac{1}{2} = e^{-\frac{t}{RC}}$$

$$\frac{1}{2} = \left(e^{-\frac{t}{RC}}\right)$$

$$\ln\left(\frac{1}{2}\right) = \ln\left(e^{-\frac{t}{RC}}\right)$$

$$-\ln(2) = -\frac{t}{RC}$$

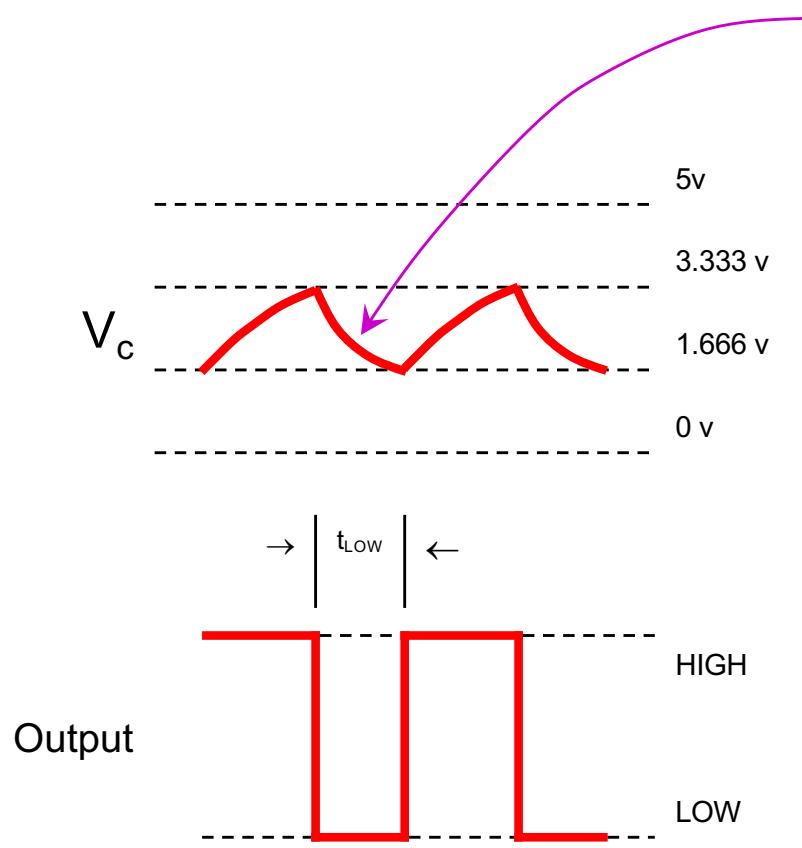
$$t_{LOW} = \ln(2) \cdot RC$$

$$t_{LOW} = 0.693 \cdot R_B C$$

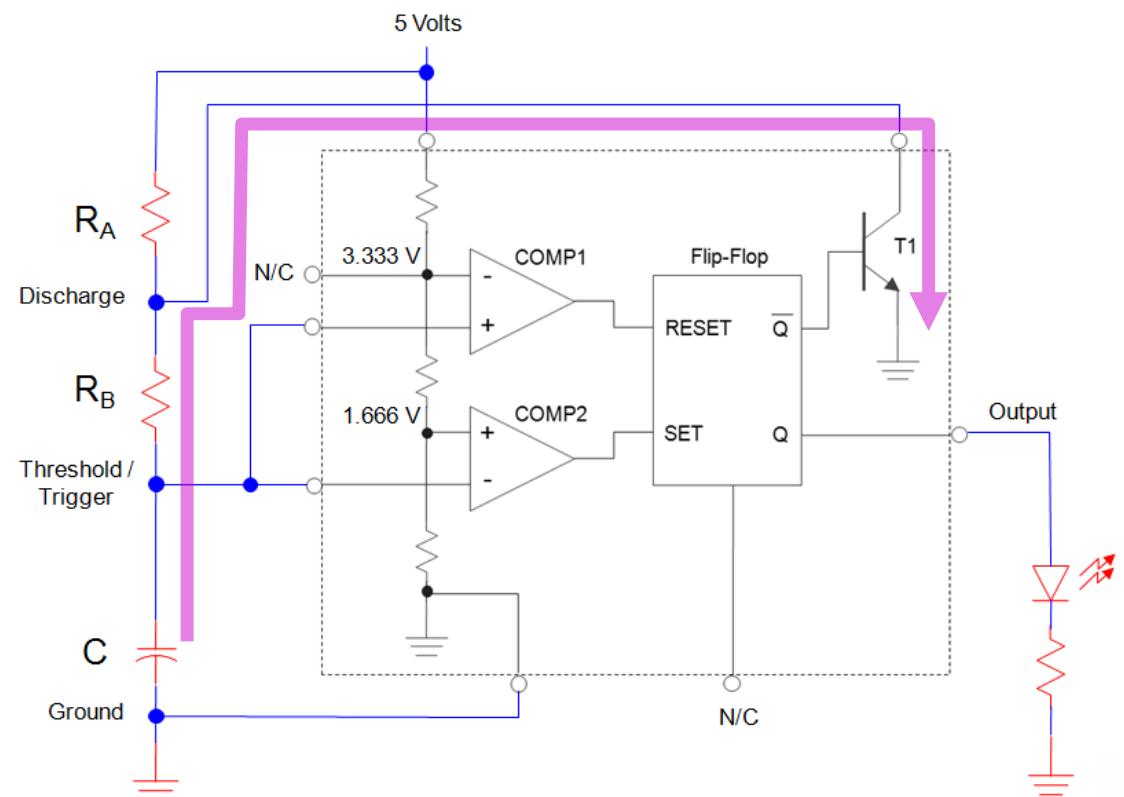
$$(\ln(2) = 0.693)$$

555 Timer Design Equations

t_{LOW} : Calculations for the Oscillator's LOW Time



THE OUTPUT IS LOW WHILE THE CAPACITOR IS
DISCHARGING THROUGH R_B .



$$t_{LOW} = 0.693 \cdot R_B C$$

Period / Frequency / Duty Cycle

Period:

$$t_{HIGH} = 0.693 (R_A + R_B) C$$

$$t_{LOW} = 0.693 R_B C$$

$$T = t_{HIGH} + t_{LOW}$$

$$T = [0.693 (R_A + R_B) C] + [0.693 R_B C]$$

$$T = 0.693 (R_A + 2R_B) C$$

$$T = (\ln 2)(R_A + 2R_B) C$$

Duty Cycle:

$$DC = \frac{t_{HIGH}}{T} \times 100\%$$

$$DC = \frac{0.693 (R_A + R_B) C}{0.693 (R_A + 2R_B) C} \times 100\%$$

$$DC = \frac{(R_A + R_B)}{(R_A + 2R_B)} \times 100\%$$

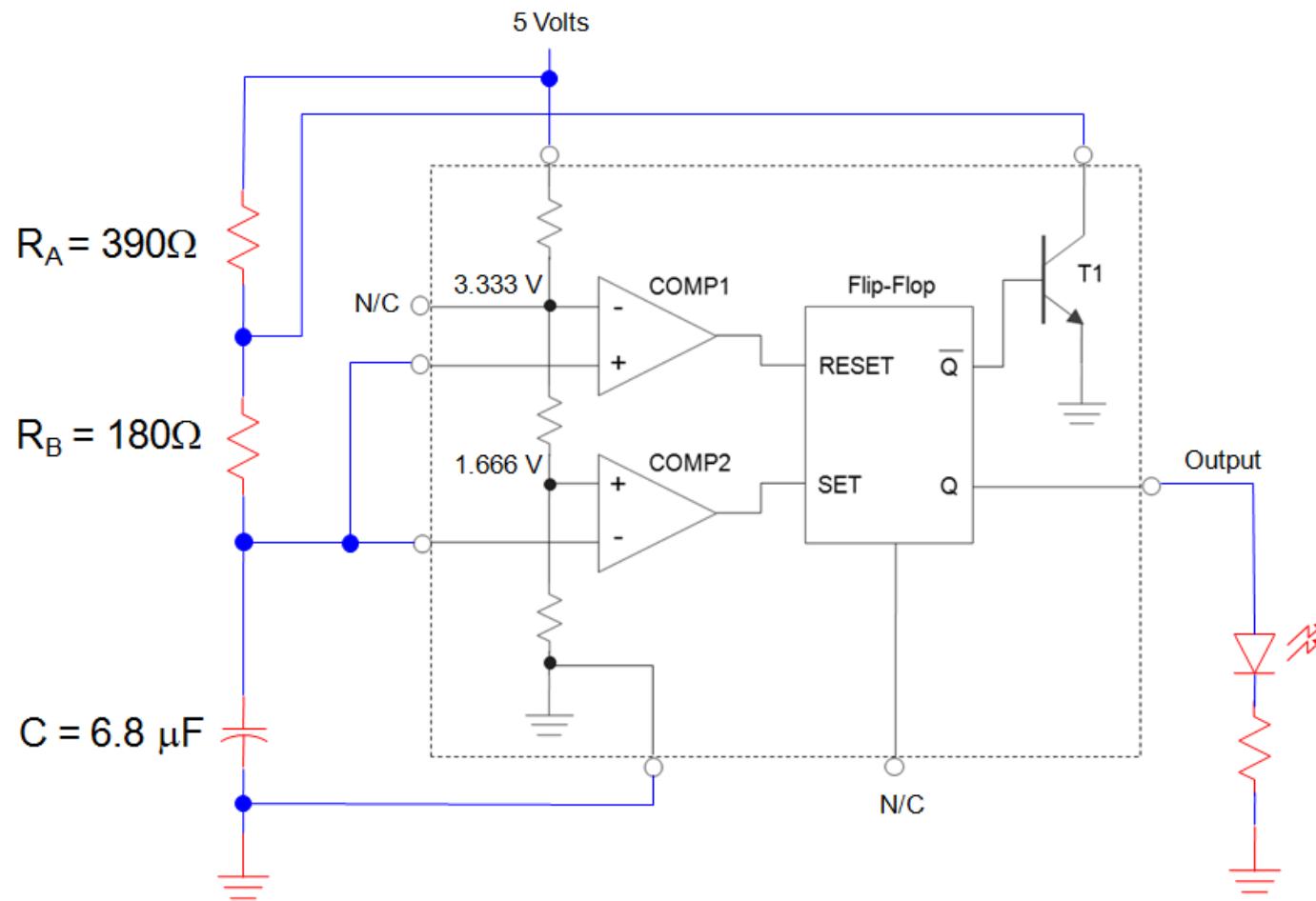
Frequency: $F = \frac{1}{T}$

$$F = \frac{1}{0.693 (R_A + 2R_B) C}$$

$$F = \frac{1}{(\ln 2) (R_A + 2R_B) C}$$

Oscillator Example 1

For the 555 Timer oscillator shown below, calculate the circuit's period (T), frequency (F), and duty cycle (DC).



Example 1: Solution

Solution:

$$R_A = 390 \Omega \quad R_B = 180 \Omega \quad C = 6.8 \mu\text{F}$$

Period:

$$T = 0.693 (R_A + 2R_B) C$$

$$T = 0.693 (390\Omega + 2 \times 180\Omega) \times 6.8\mu\text{F}$$

$$T = 3.534 \text{ mSec}$$

Frequency:

$$F = \frac{1}{T}$$

$$F = \frac{1}{3.534 \text{ mSec}}$$

$$F = 282.941 \text{ Hz}$$

Duty Cycle:

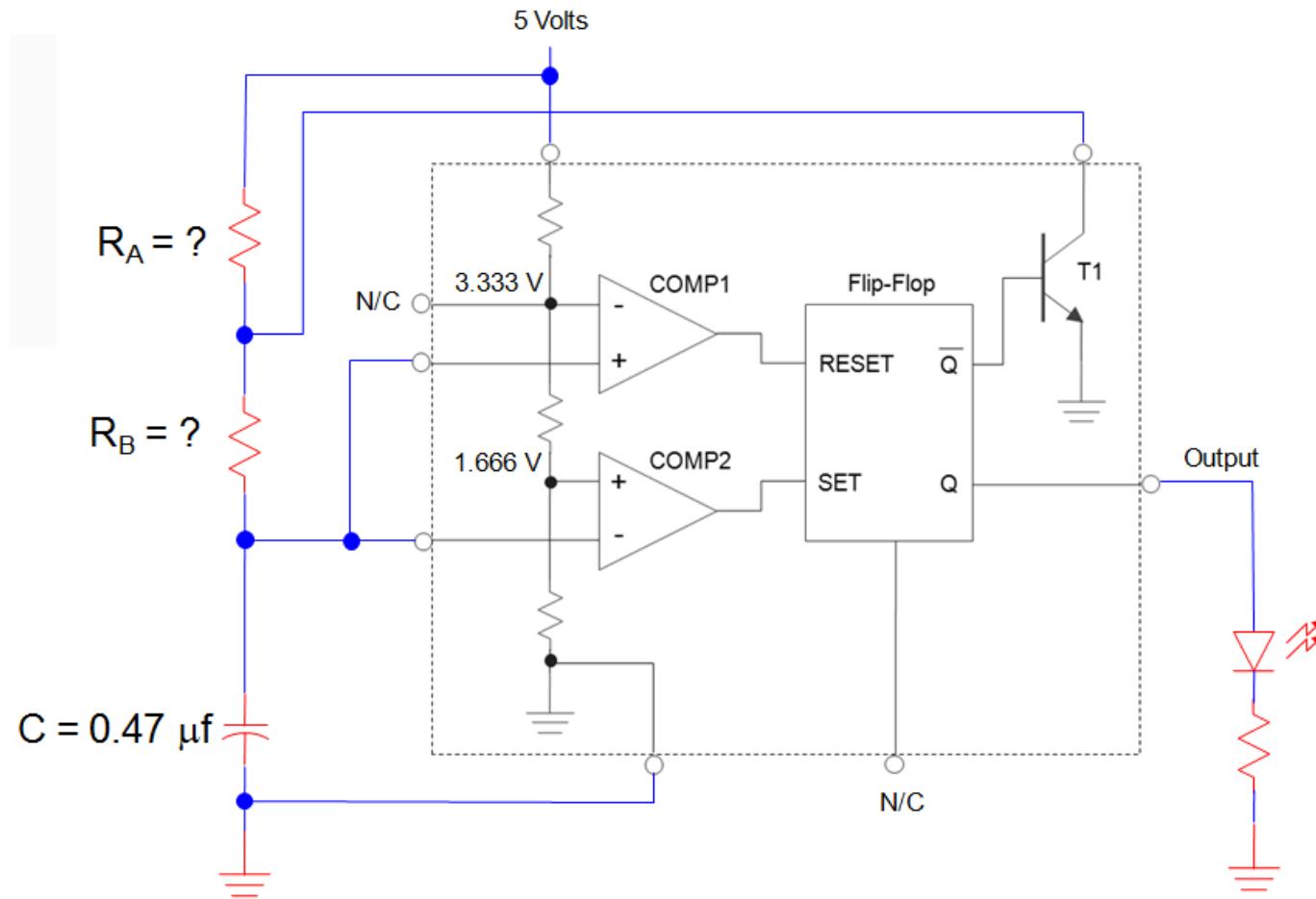
$$DC = \frac{(R_A + R_B)}{(R_A + 2R_B)} \times 100\%$$

$$DC = \frac{(390 \Omega + 180 \Omega)}{(390 \Omega + 2 \times 180 \Omega)} \times 100\%$$

$$DC = 76\%$$

Oscillator Example 2

For the 555 Timer oscillator shown below, calculate the value for R_A & R_B so that the oscillator has a frequency of 2.5 KHz @ 60% duty cycle.



Example 2: Solution

Period:

$$T = \frac{1}{f} = \frac{1}{2.5 \text{ kHz}} = 400 \mu\text{Sec}$$

$$T = 0.693 (R_A + 2R_B) C = 400 \mu\text{Sec}$$

$$T = 0.693 (R_A + 2R_B) 0.47 \mu\text{f} = 400 \mu\text{Sec}$$

$$R_A + 2 R_B = \frac{400 \mu\text{Sec}}{0.693 \times 0.47 \mu\text{f}} = 1228.09 \Omega$$

$$R_A + 2 R_B = 1228.09$$

Duty Cycle:

$$DC = \frac{(R_A + R_B)}{(R_A + 2R_B)} \times 100\% = 60\%$$

$$\frac{(R_A + R_B)}{(R_A + 2R_B)} = 0.6$$

$$R_A + R_B = 0.6(R_A + 2R_B)$$

$$R_A + R_B = 0.6 \times R_A + 1.2 \times R_B$$

$$0.4 \times R_A = 0.2 \times R_B$$

$$R_A = 0.5 \times R_B$$

Solve Two Equations & Two Unknowns

$$R_A = 245.618 \Omega$$

$$R_B = 491.23 \Omega$$

Lecture 20

Basic Logic Components
Building Components from Gates

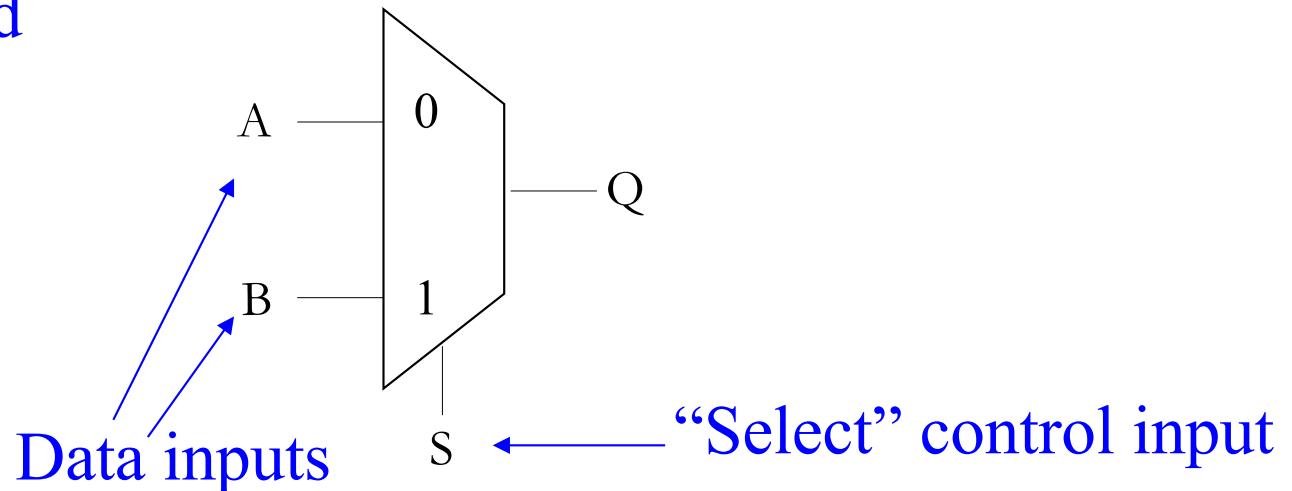
Multiplexers and Decoders

Multiplexer Circuit

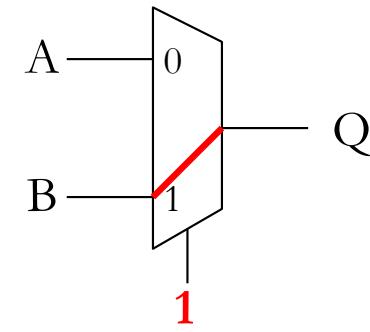
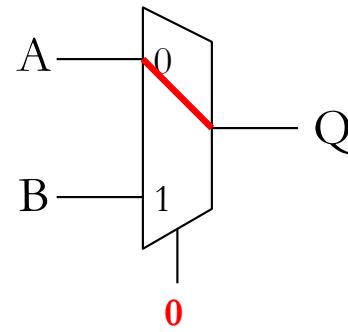
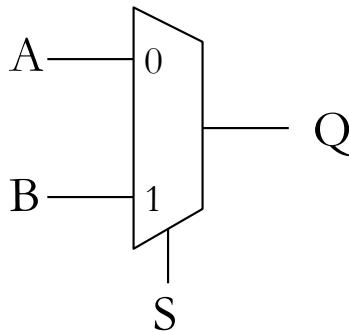
A circuit which will
select
between two data inputs (A and B) and
pass
the selected input to the output (Q).

This circuit is called
a multiplexer or
MUX for short

Symbol for a mux



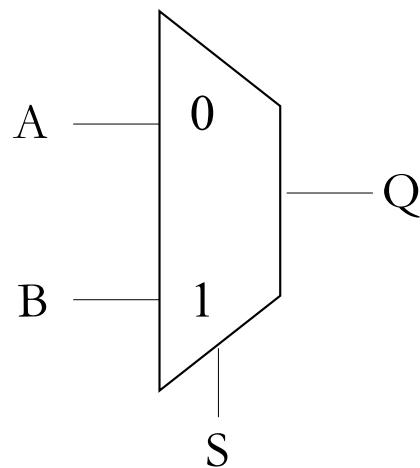
Data Steering in a Multiplexer



Key idea: The control input selects one of the inputs and passes it out to the output.

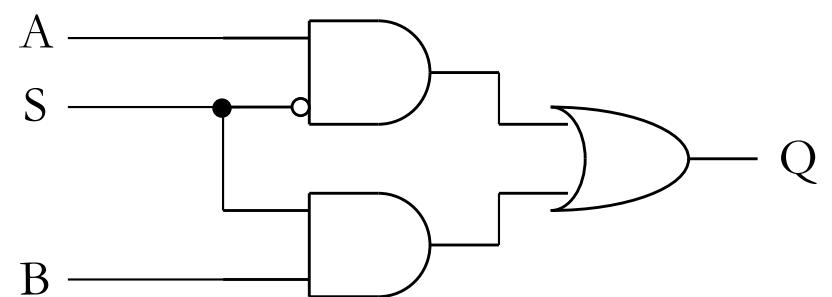
This is called a 2:1 MUX (pronounced “two-to-one”)

2-to-1 Multiplexer



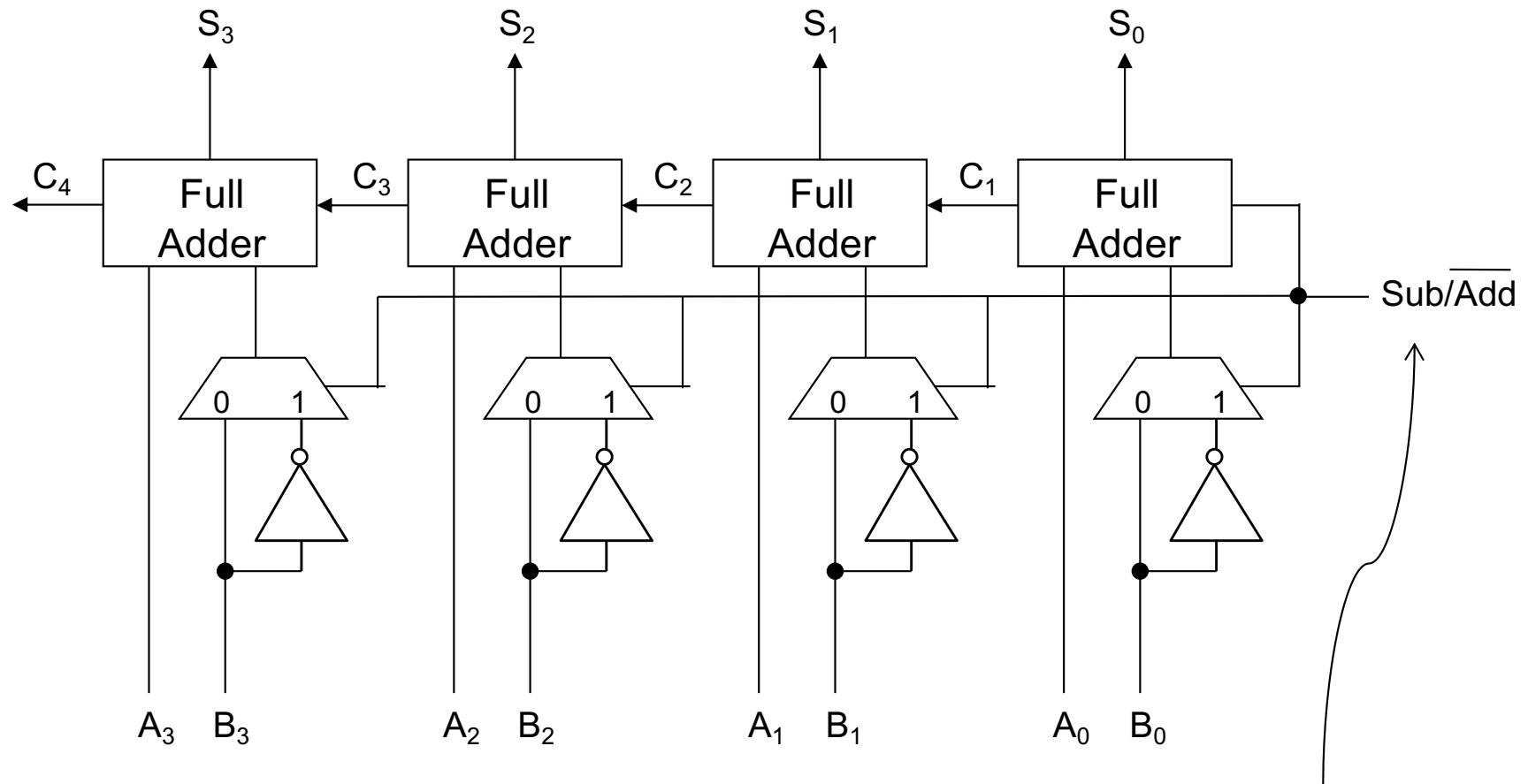
When $S=0$: $Q = A$
When $S=1$: $Q = B$

S	A	B	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



$$Q = S' A + S B$$

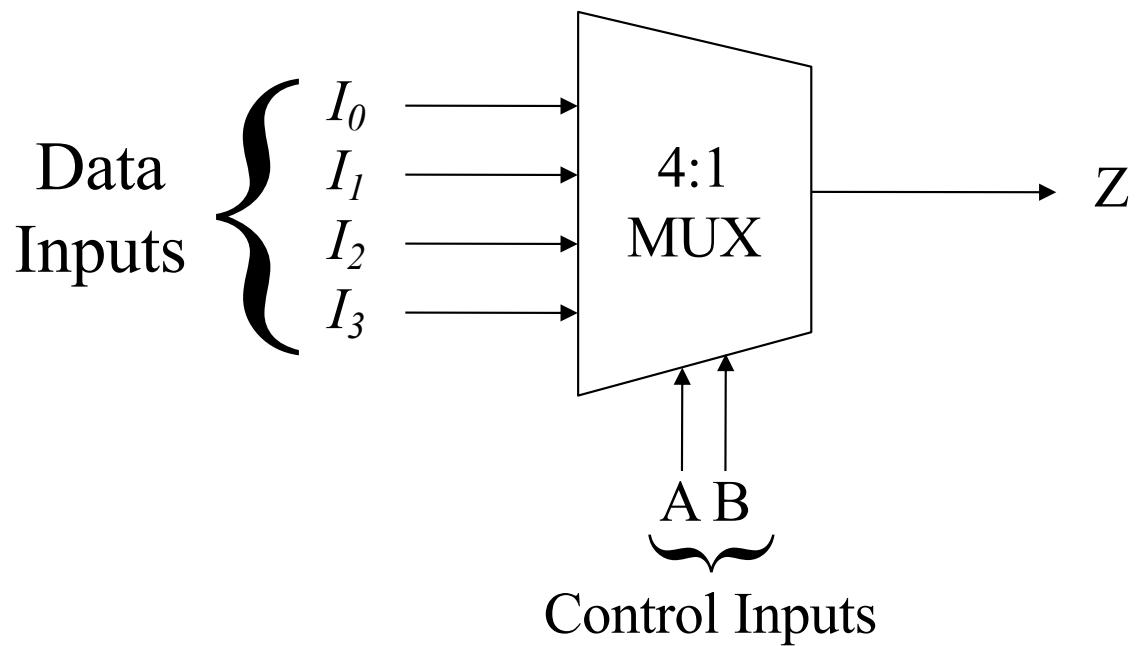
Binary Adder/Subtractor (2's complement)



Example use of a multiplexer

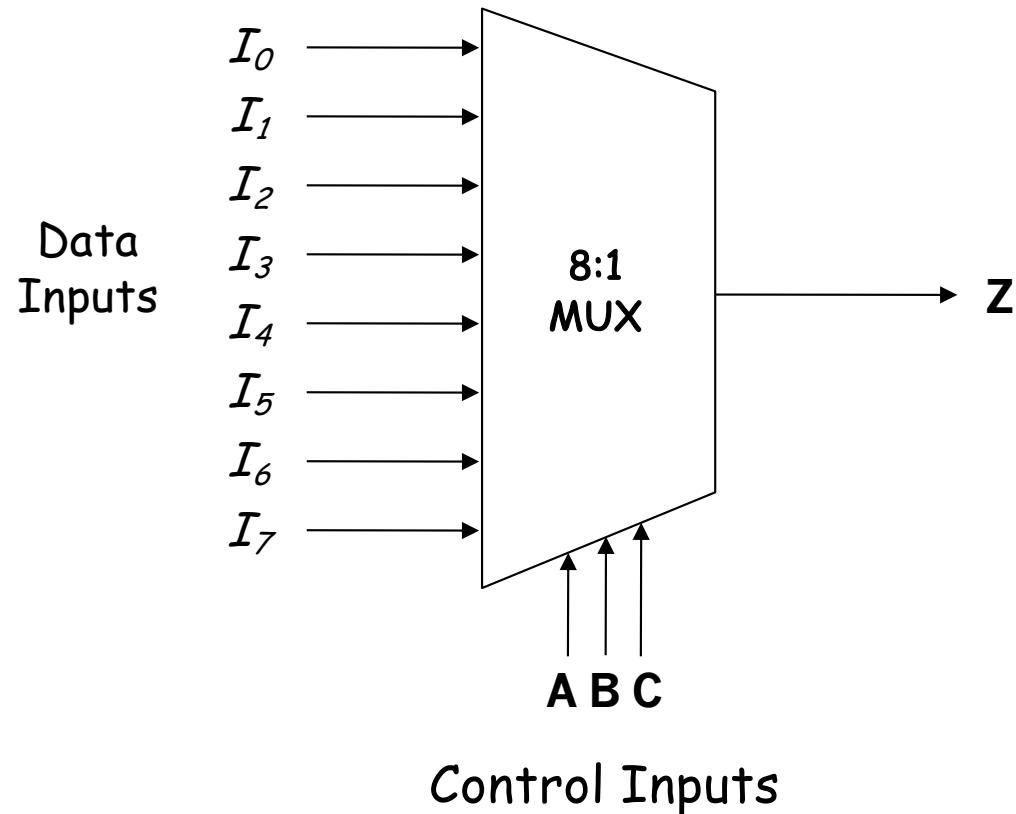
This notation means the circuit will:
 Subtract when $\text{Sub}/\overline{\text{Add}} = 1$
 Add when $\text{Sub}/\overline{\text{Add}} = 0$

4-to-1 Multiplexer



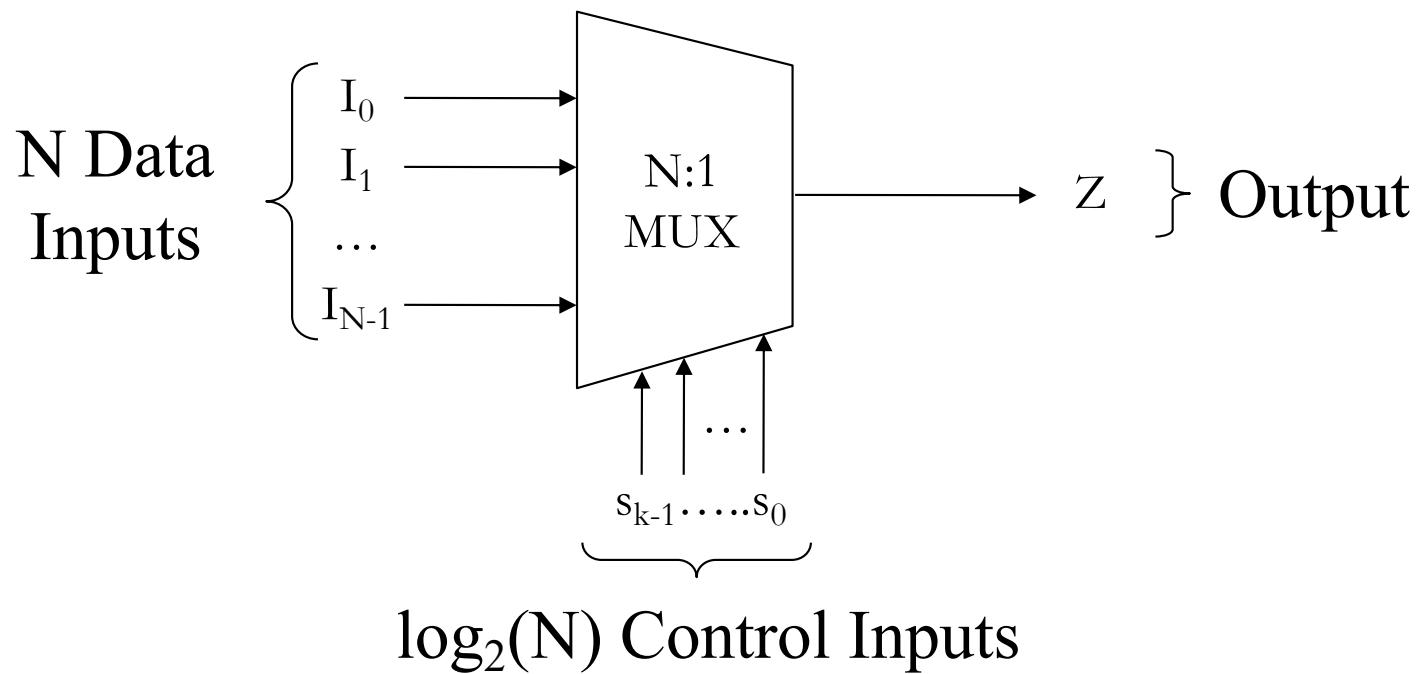
$$Z = A'B'I_0 + A'B'I_1 + AB'I_2 + ABI_3$$

8-to-1 Multiplexer

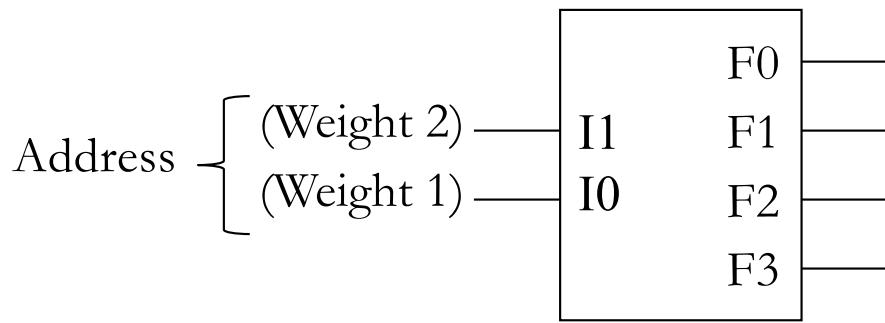


$$Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + \dots$$

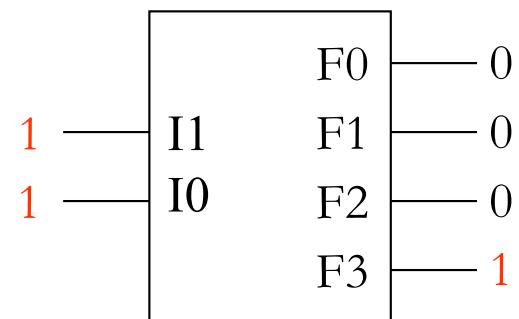
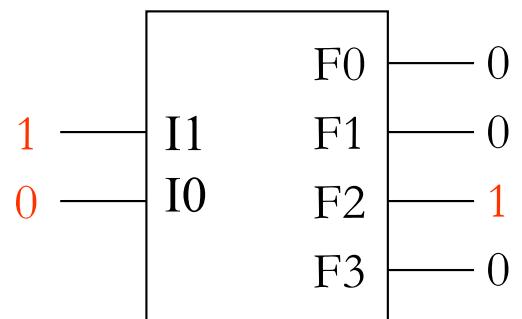
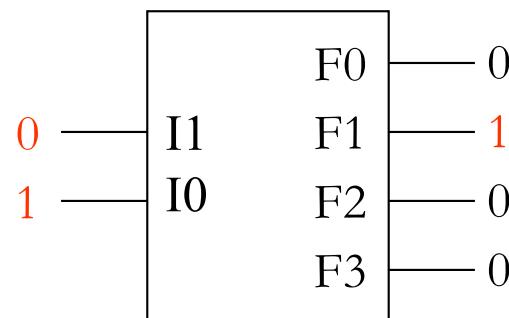
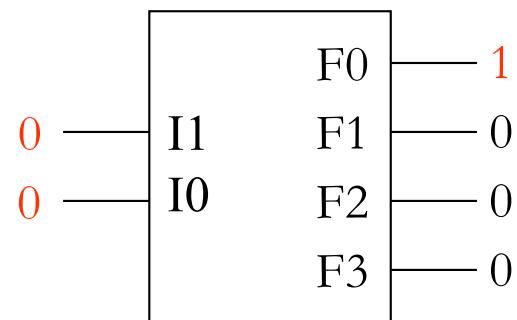
A General N-input MUX



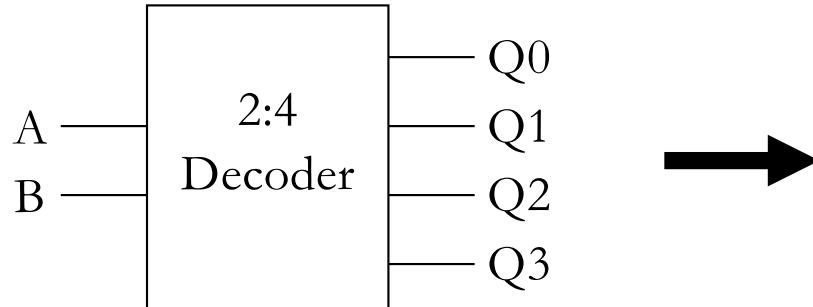
2-4 Decoder Symbol



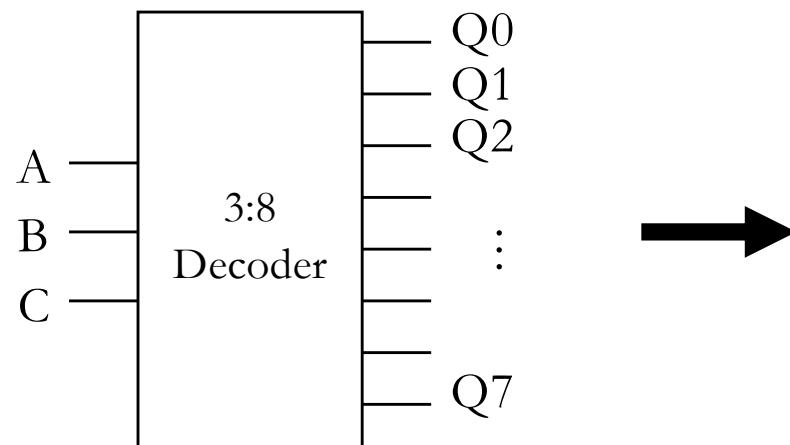
The device decodes binary weighted inputs and asserts the corresponding output.



2-to-4 and 3-to-8 Decoders



A	B	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



A	B	C	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

In general, a decoder has n inputs and 2^n outputs

Memory Circuits

Set-Reset Flip-Flops

Gated D-Latch

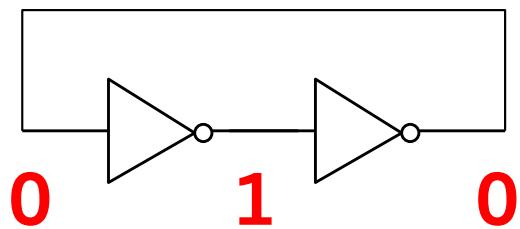
D Flip-Flops

Clocks

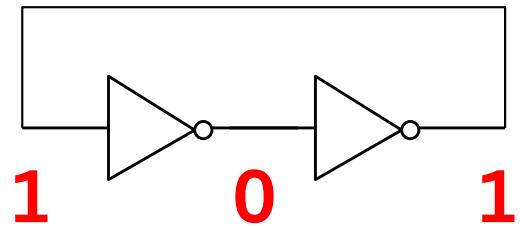
Registers

Bi-Stability

The Key to Memory



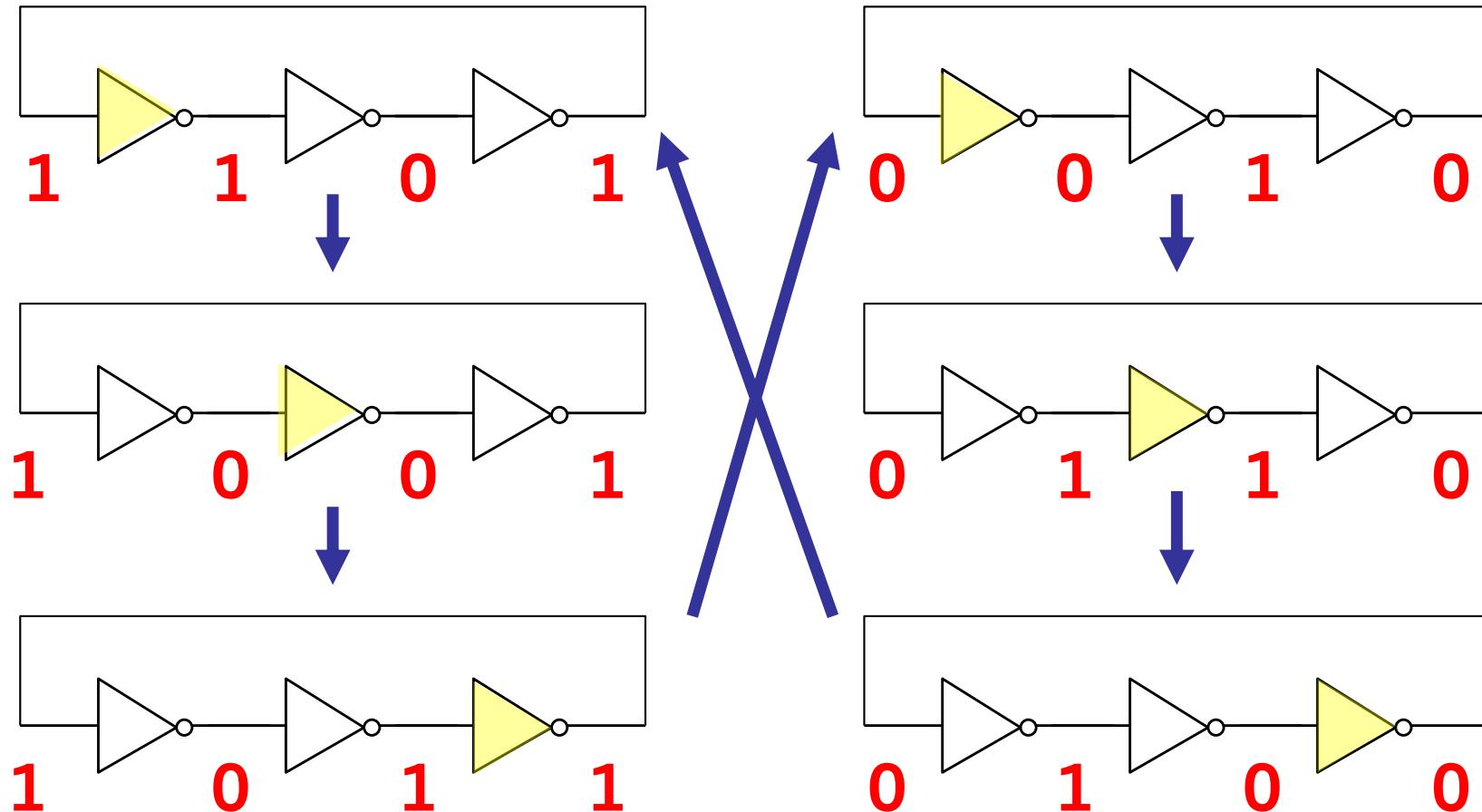
This is a stable state –
it will sit like this forever
(or until the power is turned off)



This is also a stable state –
it will sit like this forever
(or until the power is turned off)

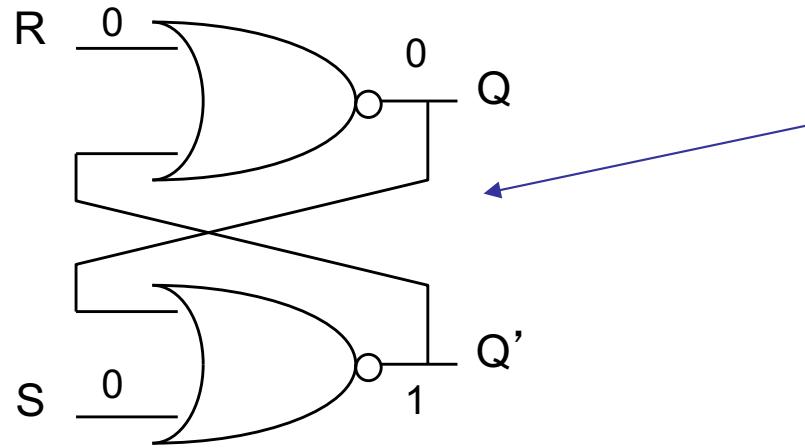
There are 2 stable states –
a **bi-stable** circuit...

An Unstable Circuit

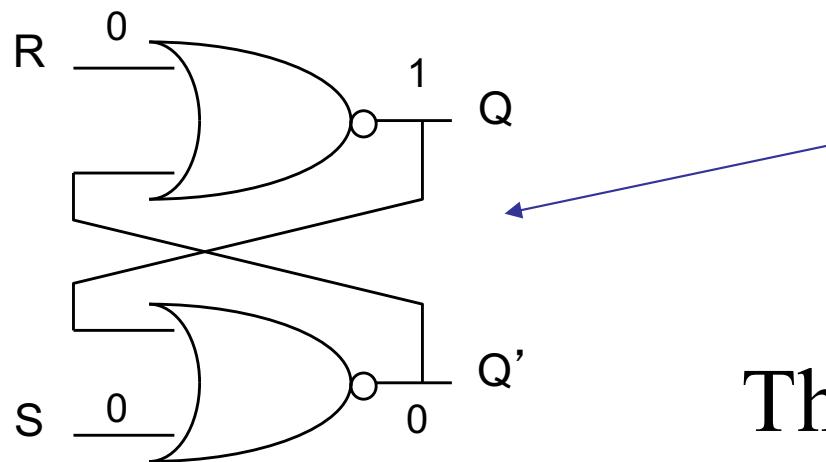


This circuit is unstable. It oscillates.

Set-Reset Flip-Flop – A Bi-Stable Circuit



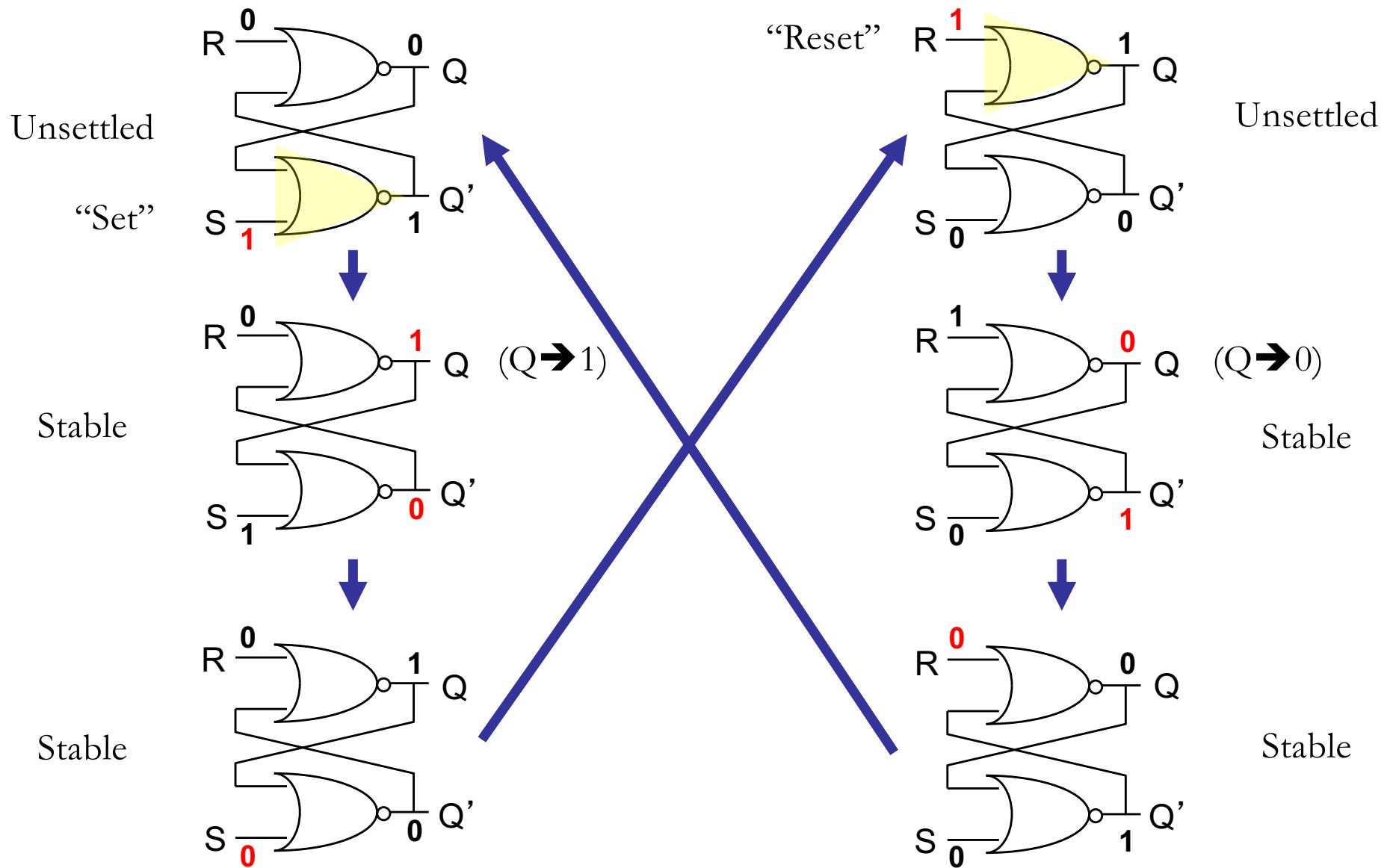
This is a stable state –



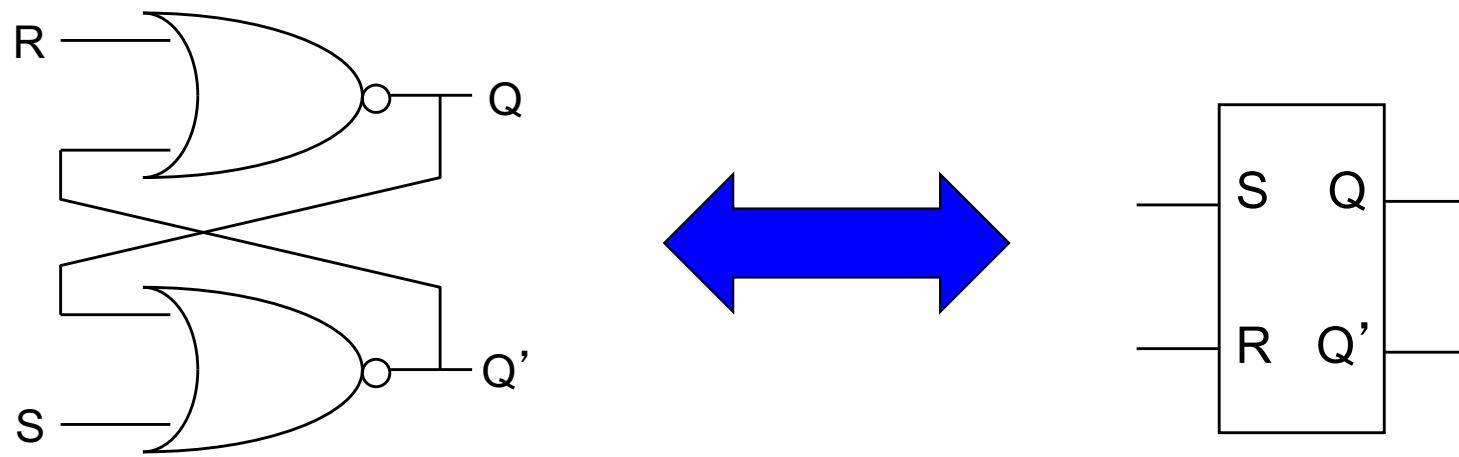
This is also a stable state –

There are 2 stable states –
a **bi-stable** circuit...

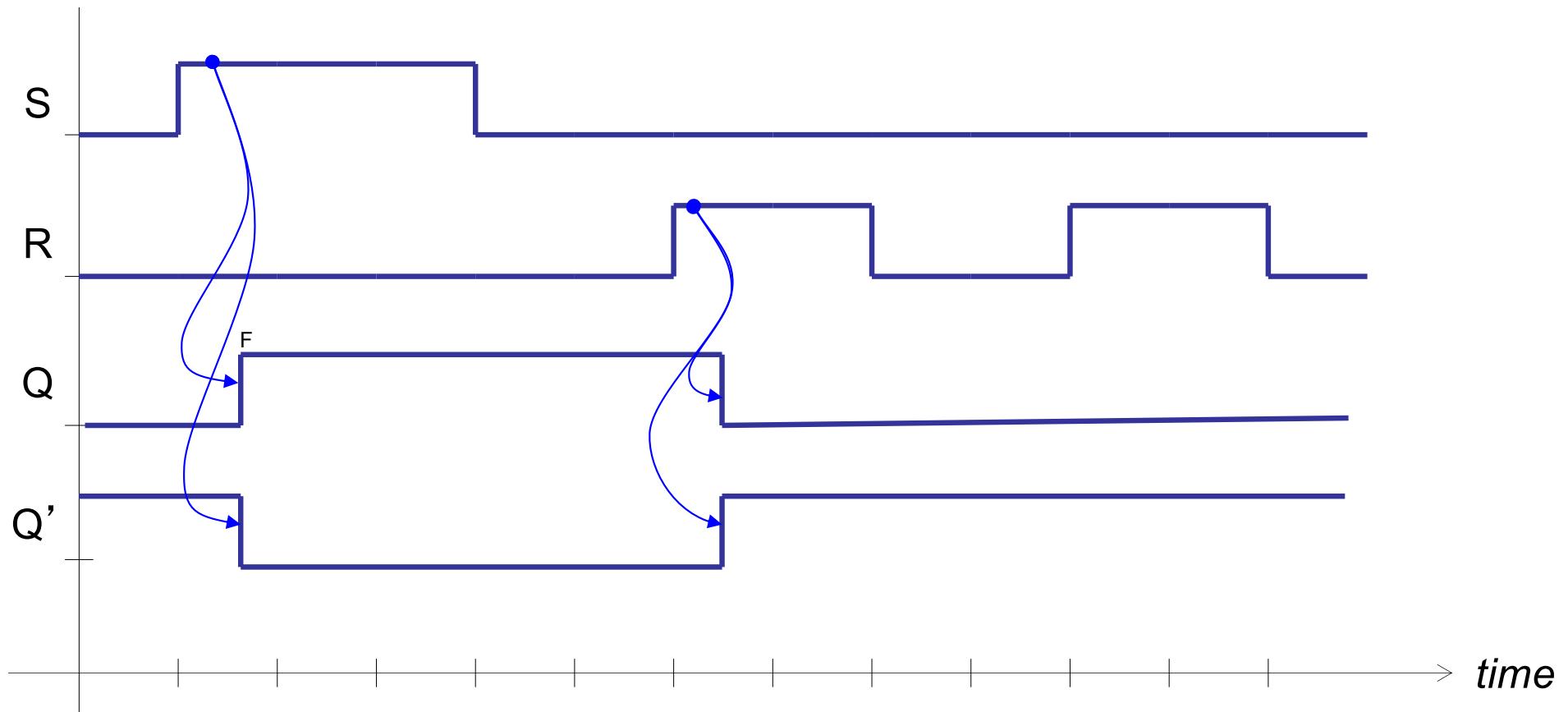
Set-Reset Flip-Flop Transitions



Set-Reset Flip-Flop Symbology



Set-Reset Flip-Flop Timing Diagram

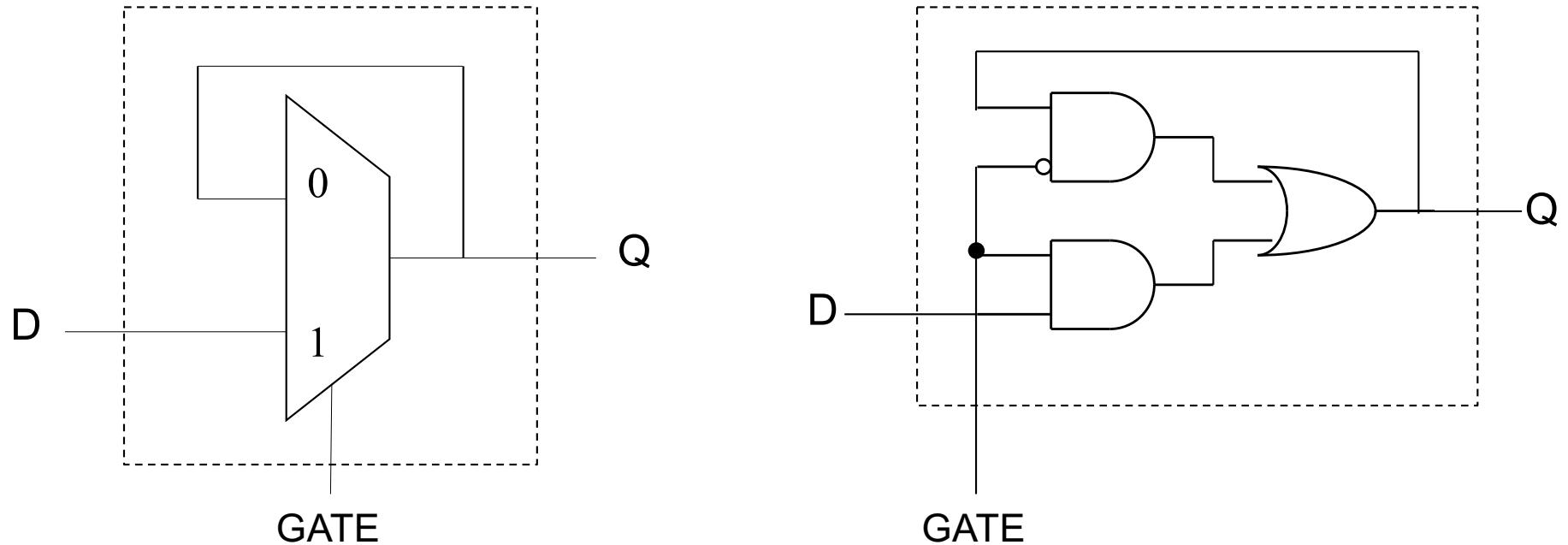


- Diagrams the logic behavior of digital circuits as a function of time
- Show causality (the cause-effect relations) between signals

The Power of SR Flip-Flops

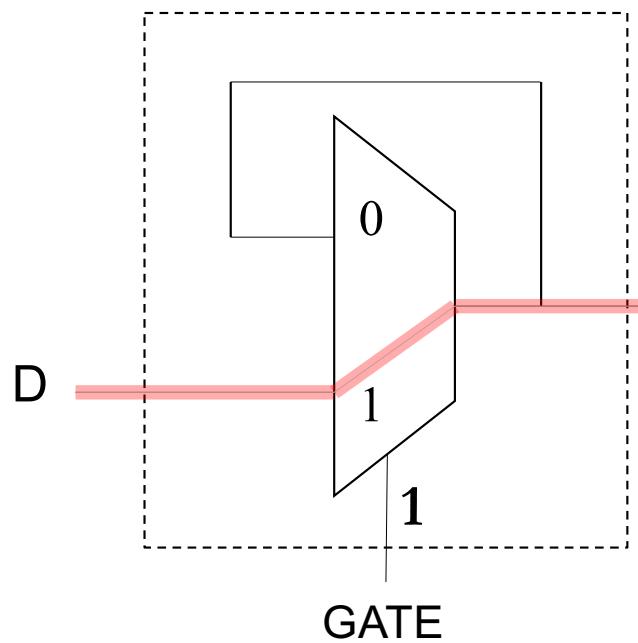
- Illustrate the simple notion of *bi-stability*
 - Two stable states
 - S and R inputs move latch between the two states
- Function as memory
 - When $Q = 1 \Leftrightarrow$ memory storing a '1'
 - When $Q = 0 \Leftrightarrow$ memory storing a '0'
- Will hold its value indefinitely
 - As long as circuit is powered

Gated D Latch

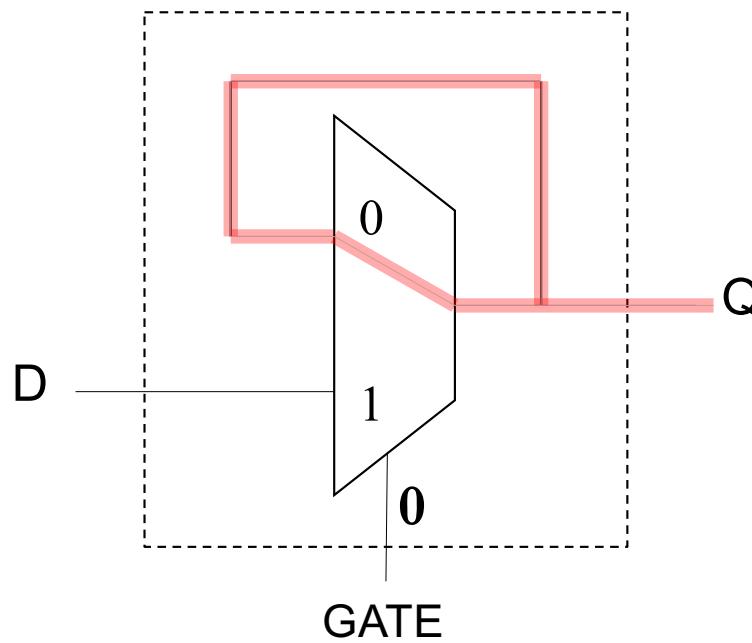
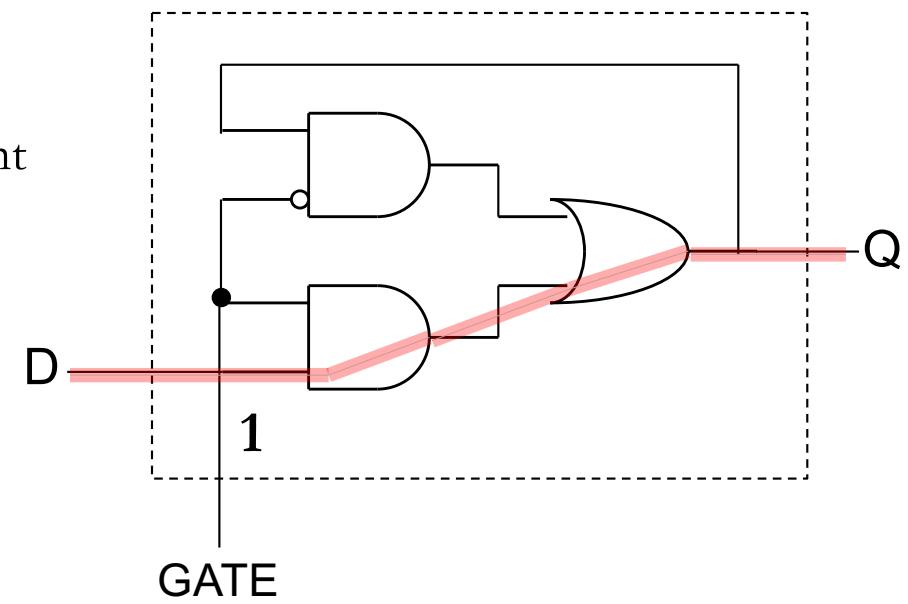


- Sometimes called a *transparent* latch
 - When GATE = 1, the D input propagates (loads) to Q output
 - When GATE = 0, Q does not change, its value is stored.
- Allows us to control *when* to store new data into latch

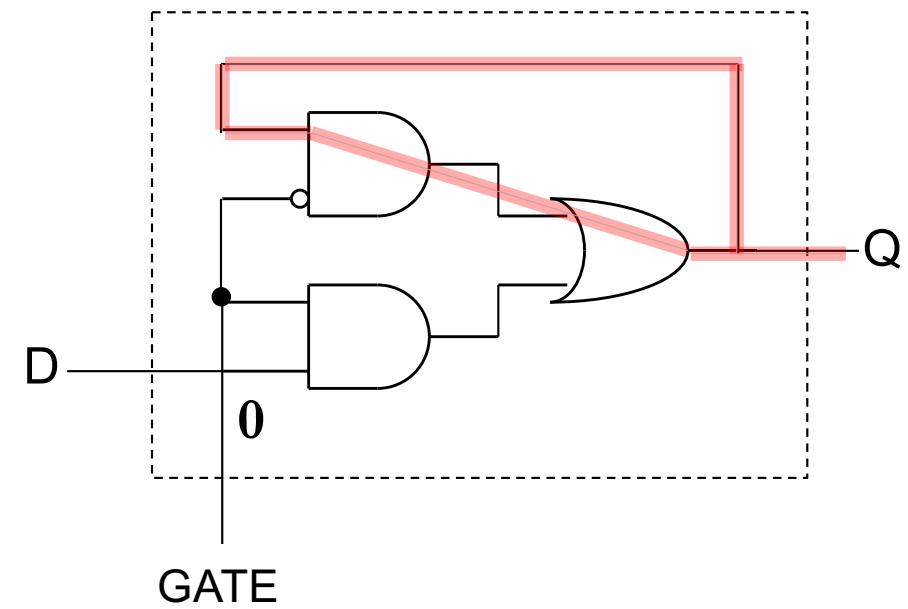
Gated D Latch



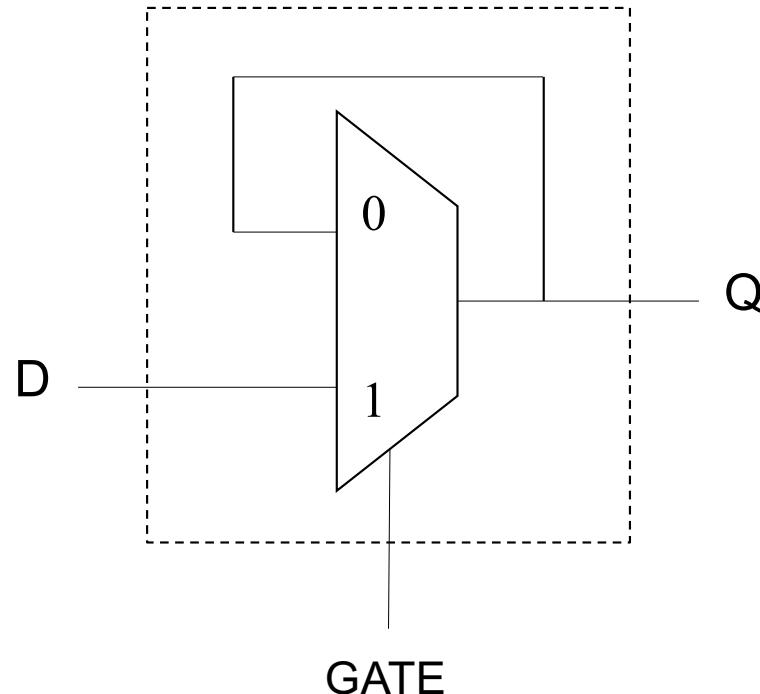
Gate = 1
Latch is
Transparent
 $Q = D$



Gate = 0
Latch is
Latched
 $Q = Q$



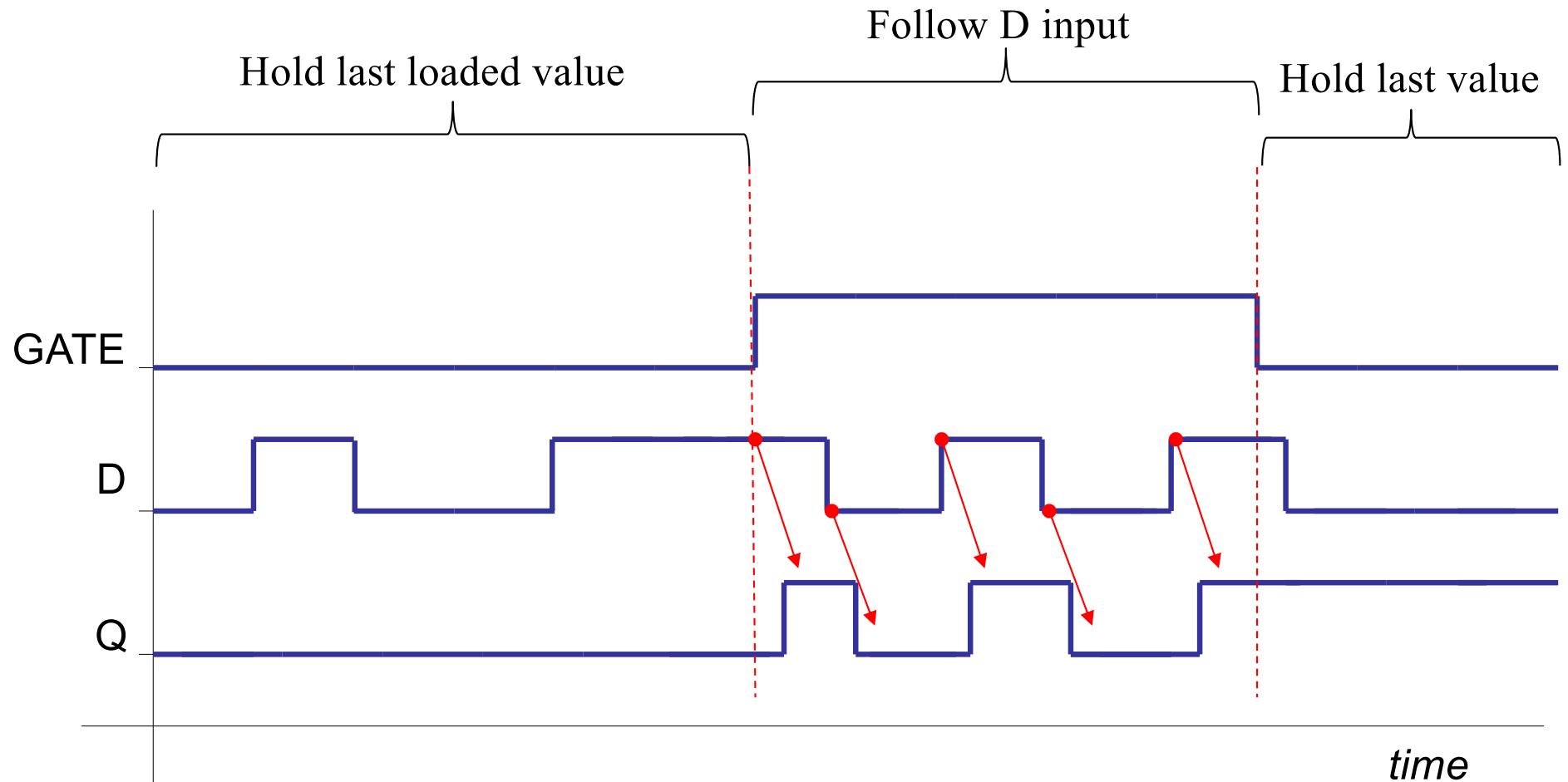
The Gated D Latch



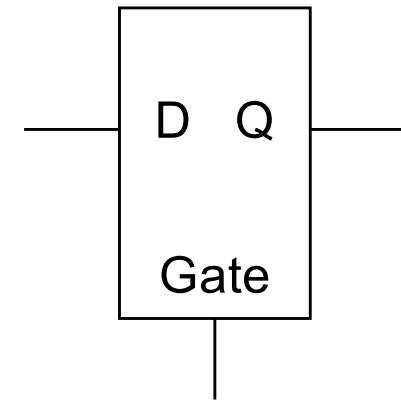
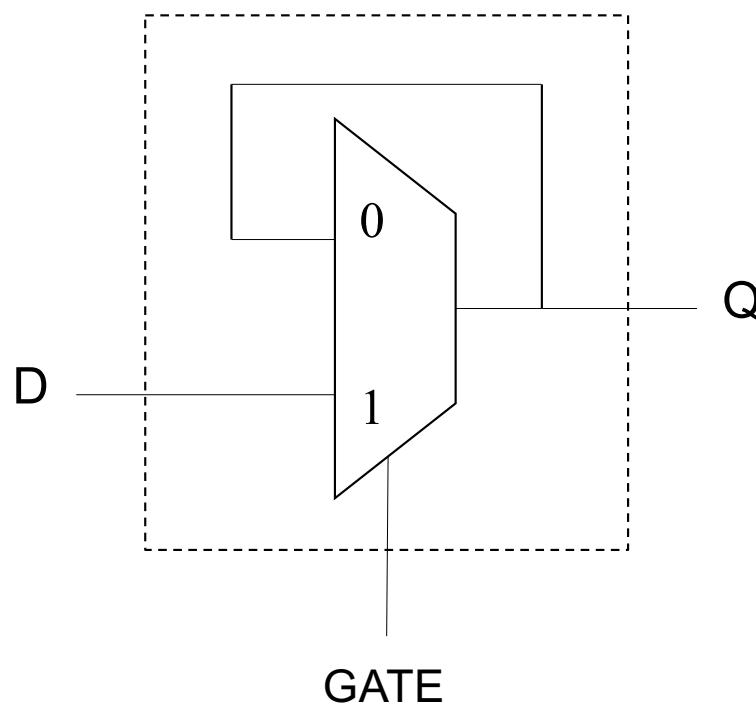
$$Q = \text{GATE} \cdot D + \text{GATE}' \cdot Q$$

When $\text{GATE} = 1 \Leftrightarrow Q \text{ follows } D \quad (\text{load})$
When $\text{GATE} = 0 \Leftrightarrow Q \text{ retains old value} \quad (\text{store})$

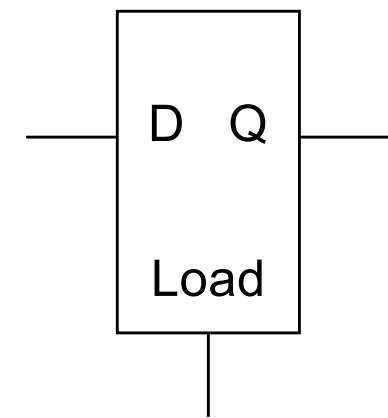
Gated D Latch – Timing Diagram



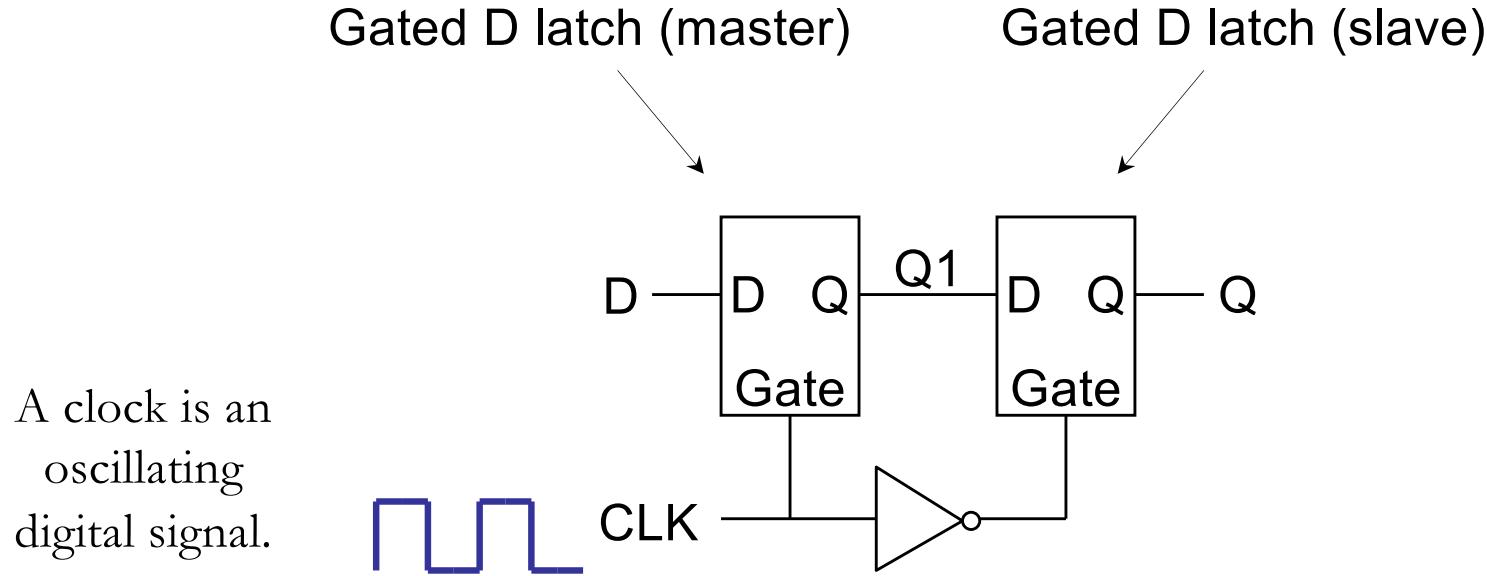
Gated D-Latch Symbology



or...



The Master/Slave Flip Flop (D Type)



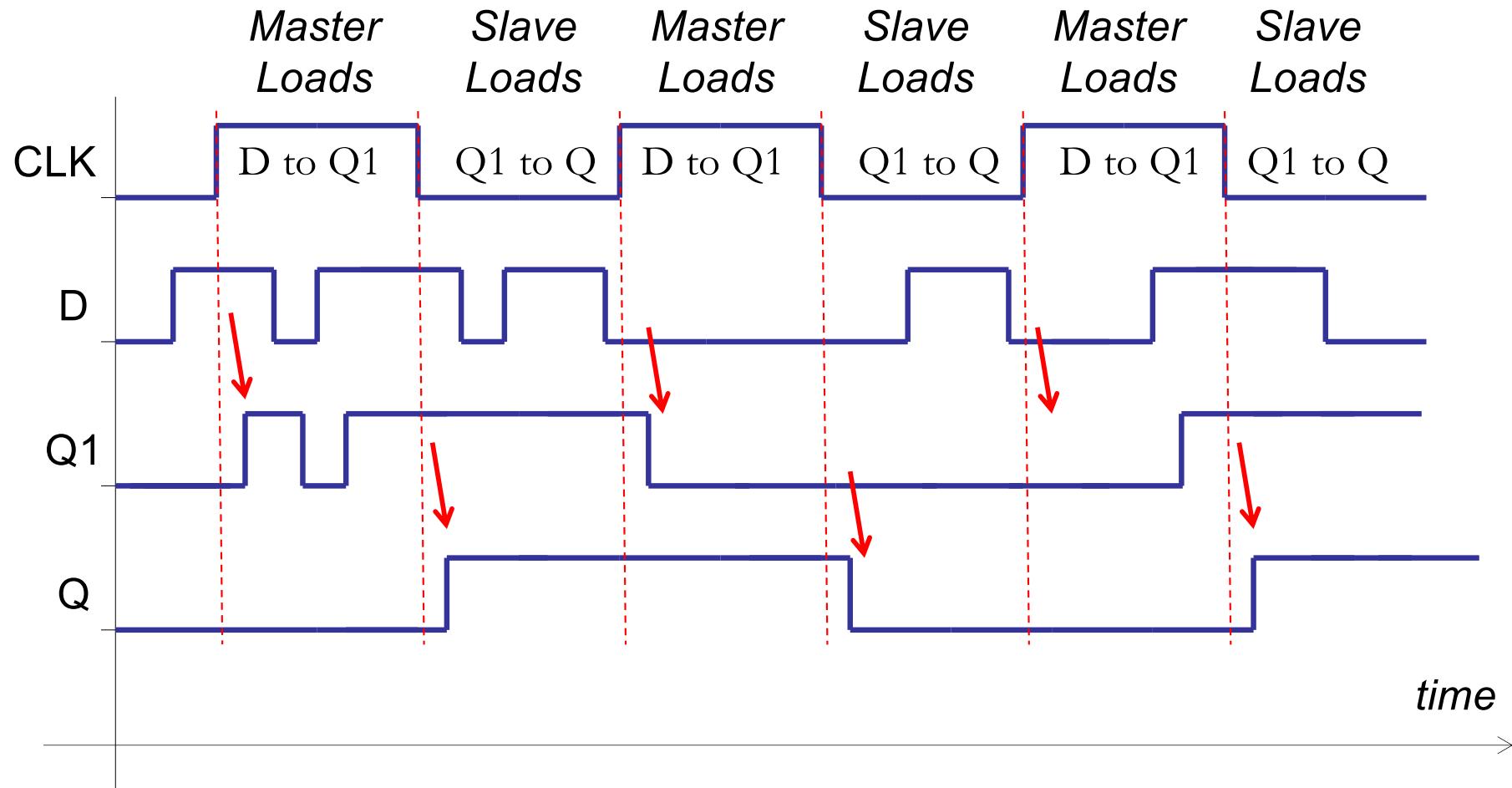
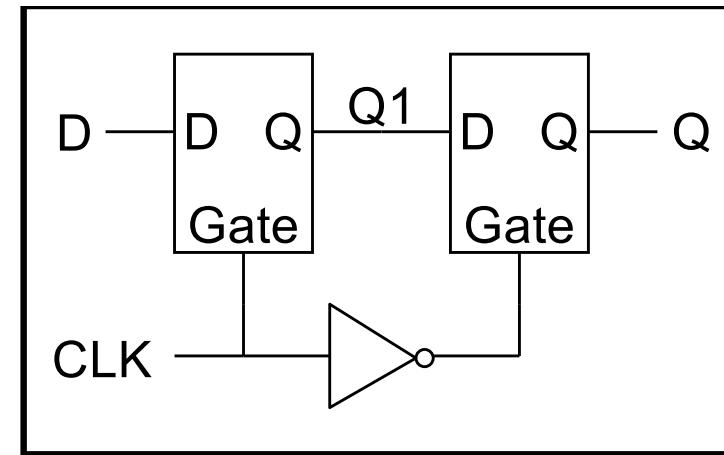
Either:

- The master is loading and the slave is holding ($\text{CLK} = 1$), or
- The slave is loading and the master is holding ($\text{CLK} = 0$).

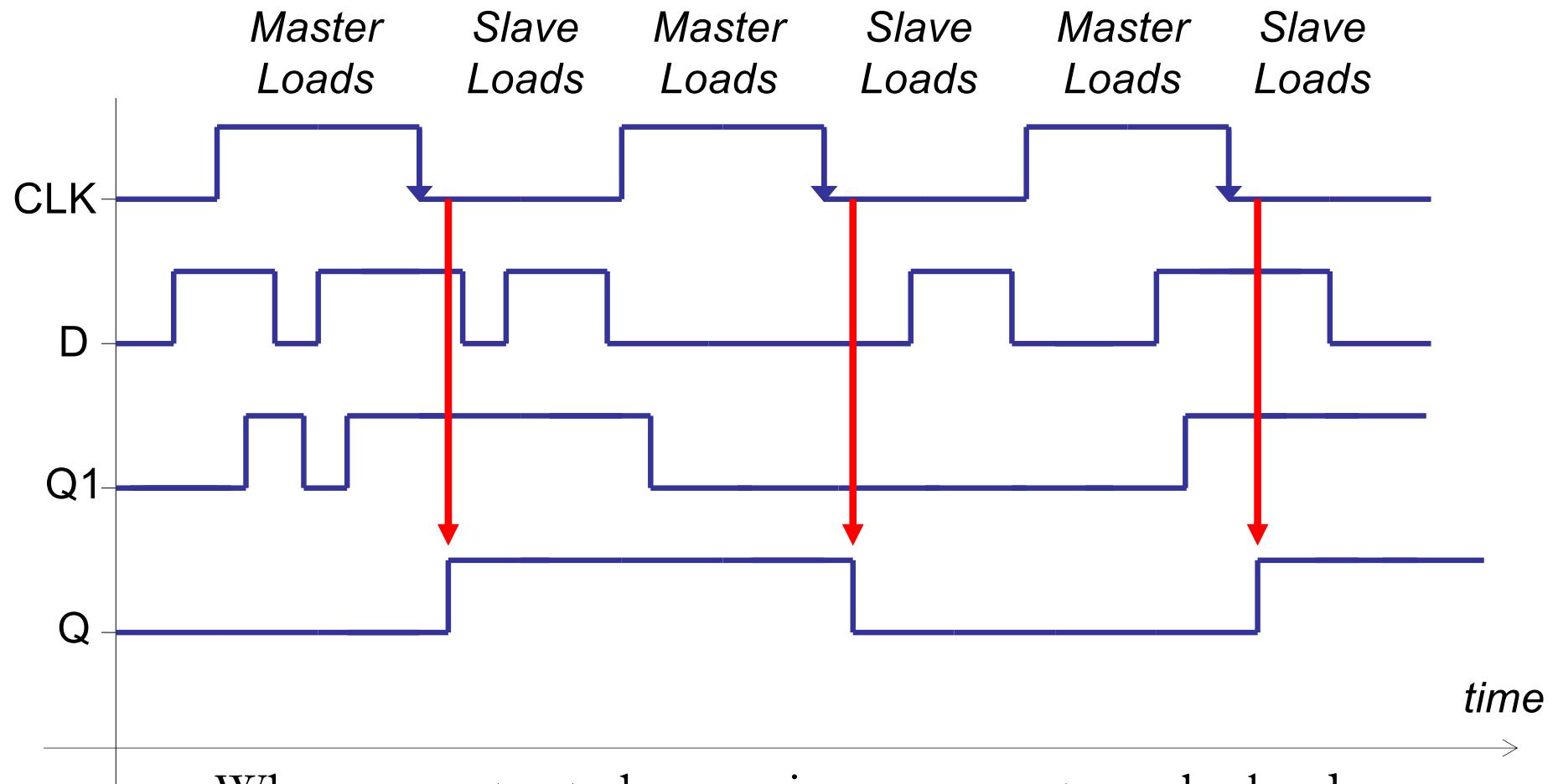
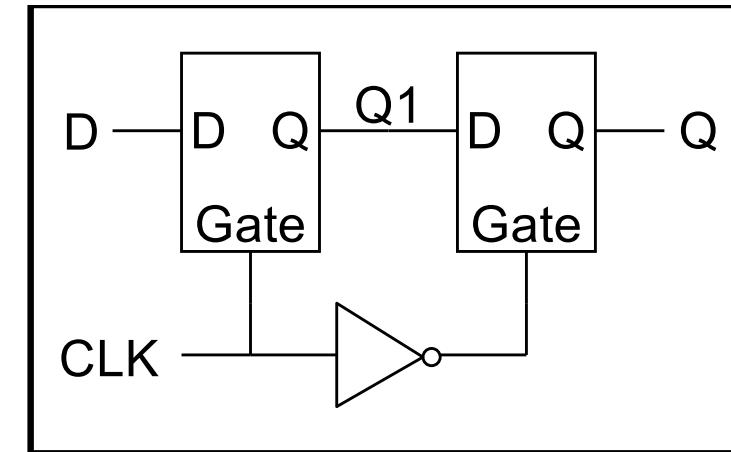
But never both at the same time...

Q only changes when CLK transitions from 1 to 0 – why.

Master-Slave D Flip-Flop Timing Diagram



Output Changes in Response to Falling Clock

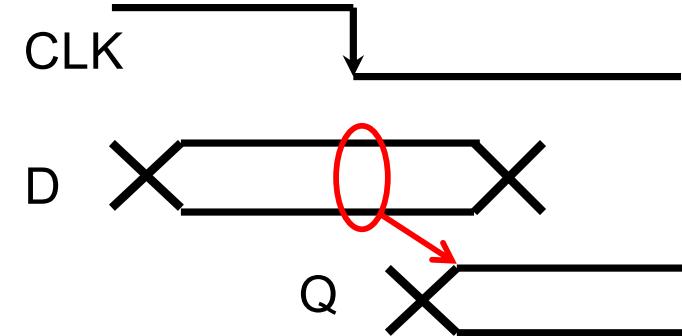


When an output changes in response to a clock edge,
it is called “Edge Triggered”

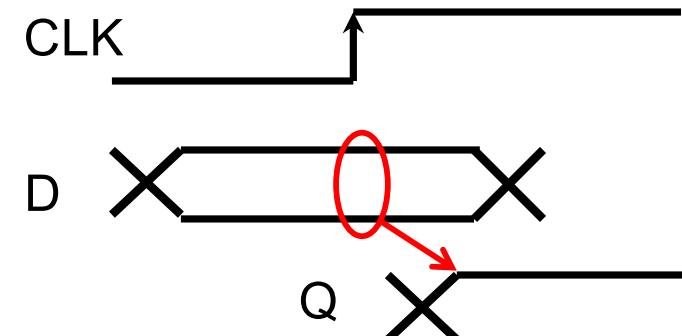
Edge Triggered D Flip-Flop Symbol



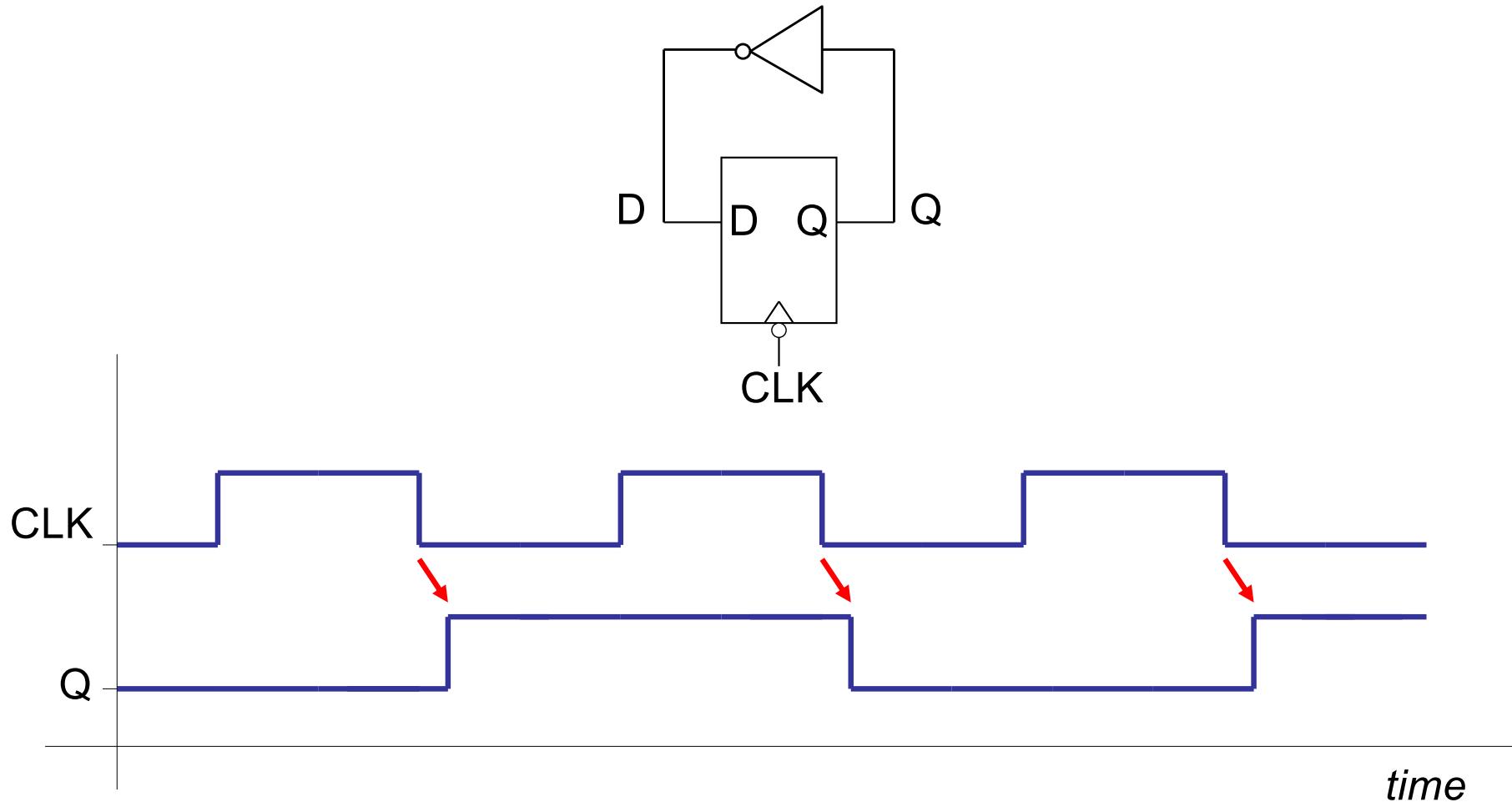
\circ Falling edge triggered



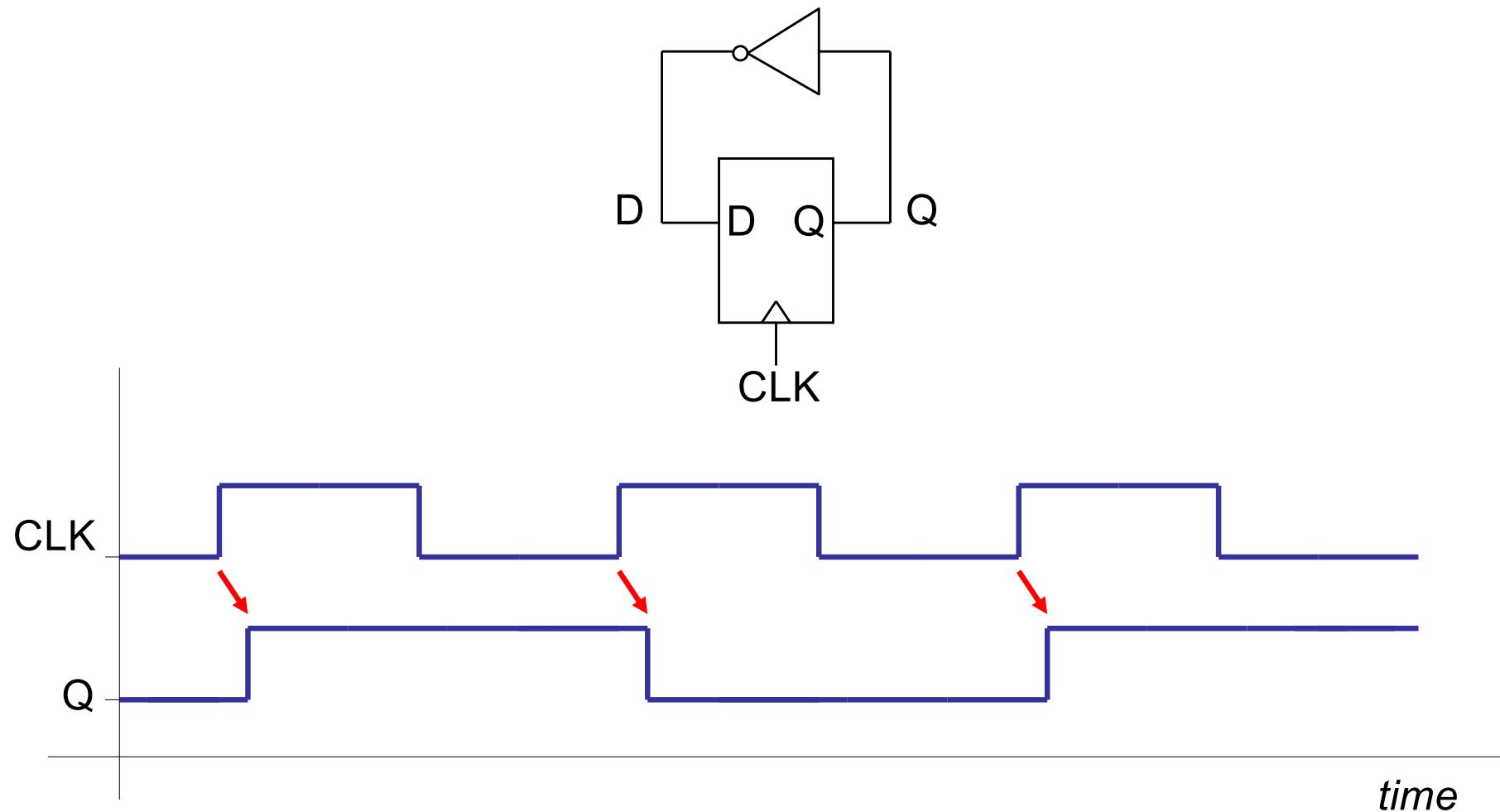
\circlearrowleft Rising edge triggered



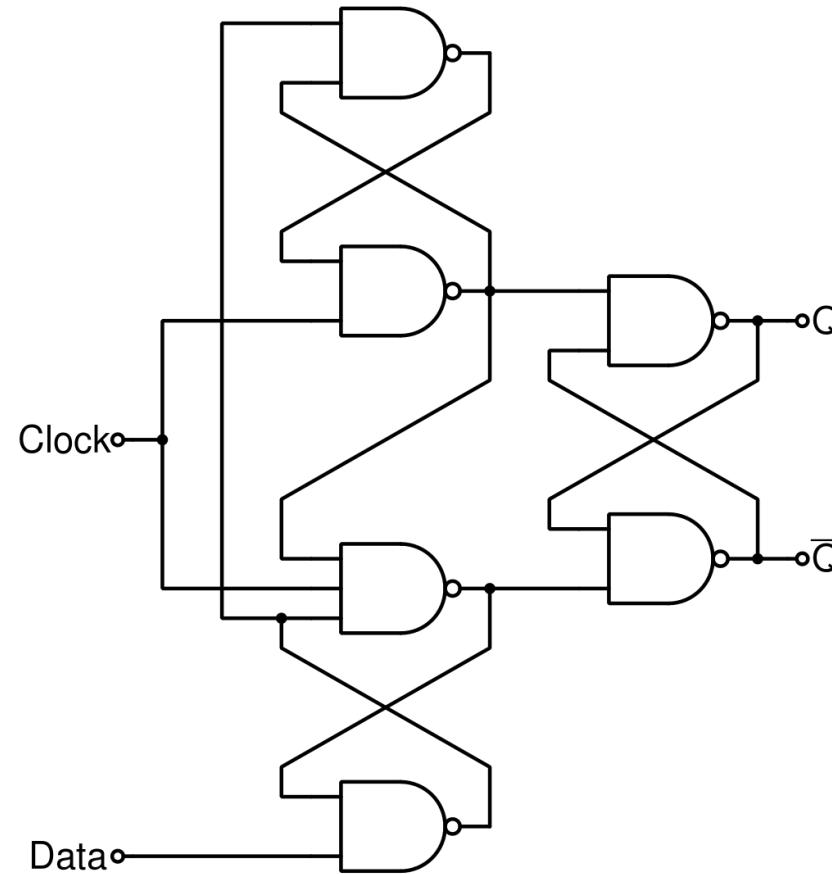
Oscillator (Toggle Circuit) Operation



Oscillator (Toggle Circuit) Operation



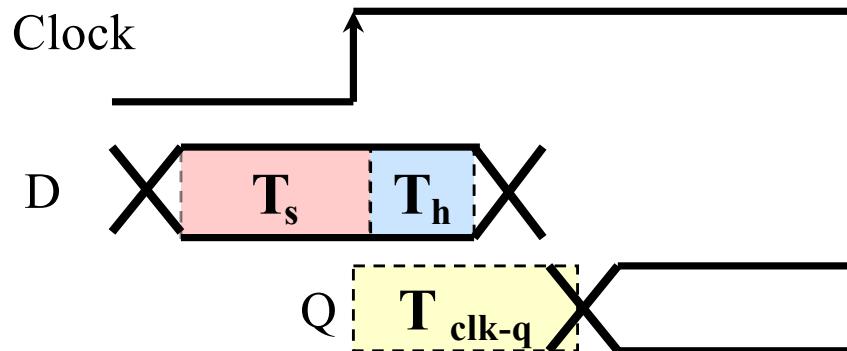
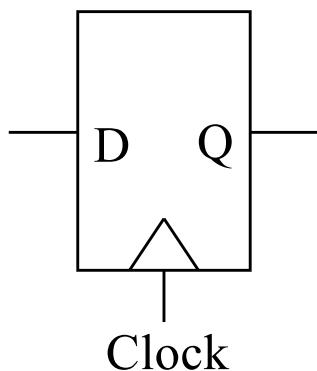
7474 D Flip-Flop



In reality, a D Flip-Flop is not implemented with a master-slave architecture. This is the usual logic circuit.
Notice it uses 3 Set-Reset flip-flops internally !

Definition of Basic D Flip-Flop Timing Parameters

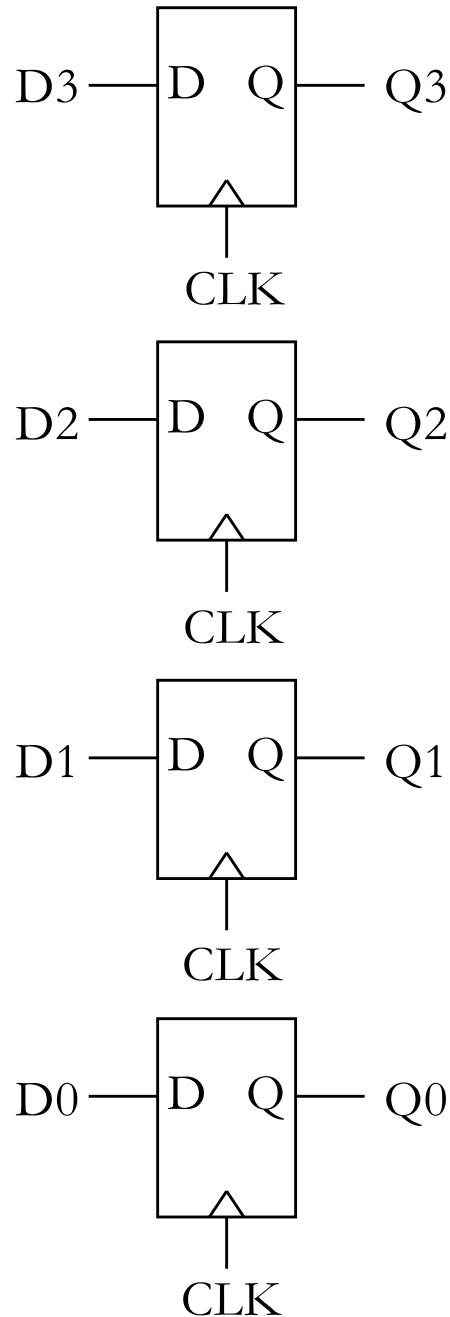
- Setup time (T_s)
 - Amount of time input D must be stable before a *Clock* event
- Hold time (T_h)
 - Amount of time input D must remain stable after a *Clock* event
- Clk-to-Q delay time (T_{clk-q})
 - delay from *Clock* event to when Q output becomes available



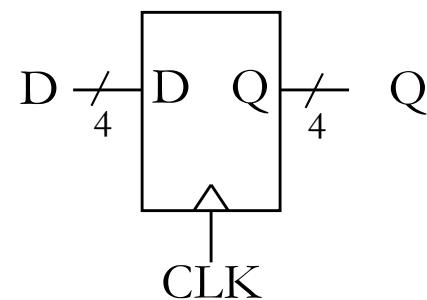
Lecture 21

Registers
Loadable Registers
Accumulators
Register File
Digital CMOS

Registers

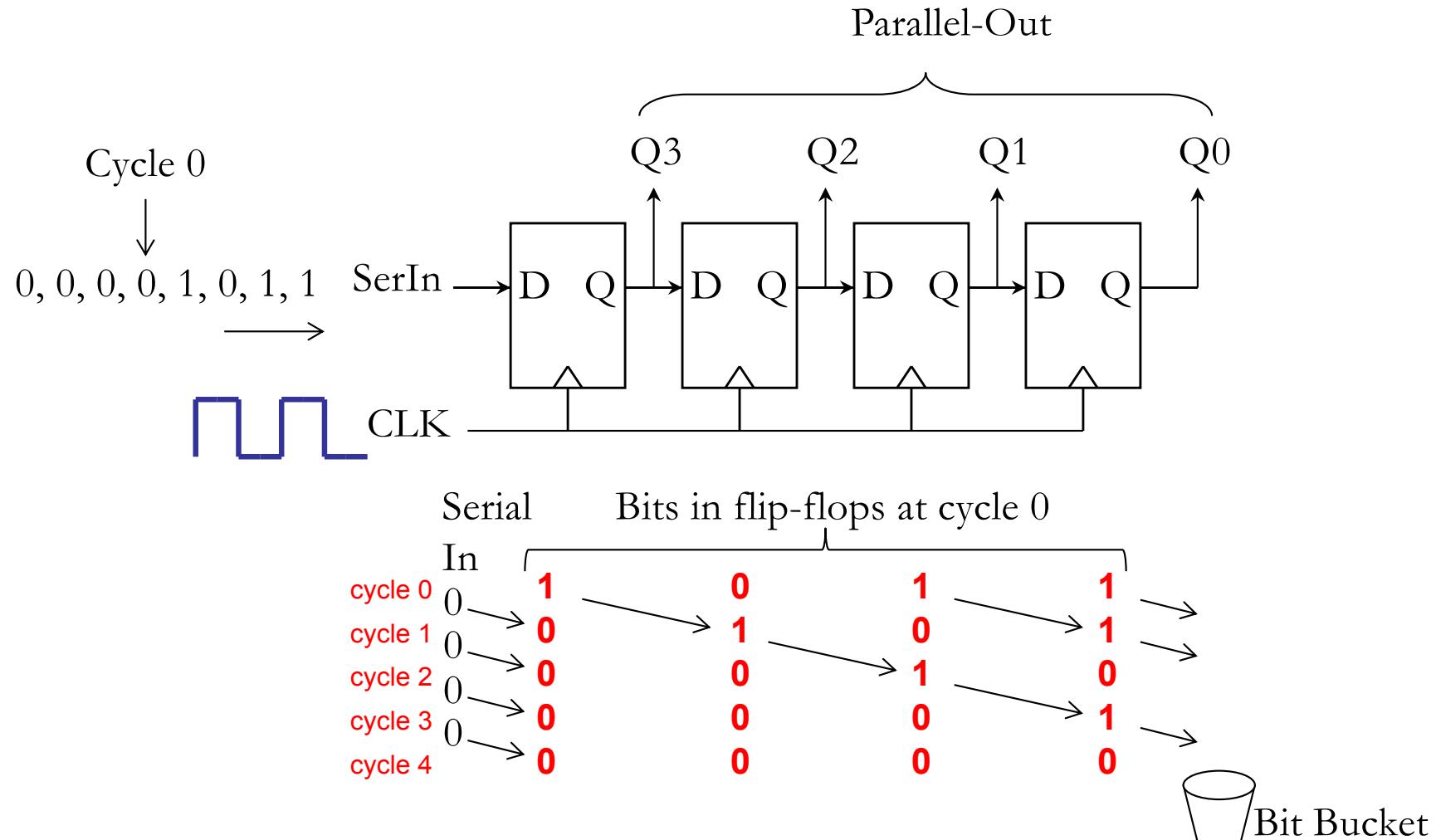


A 4-Bit Register



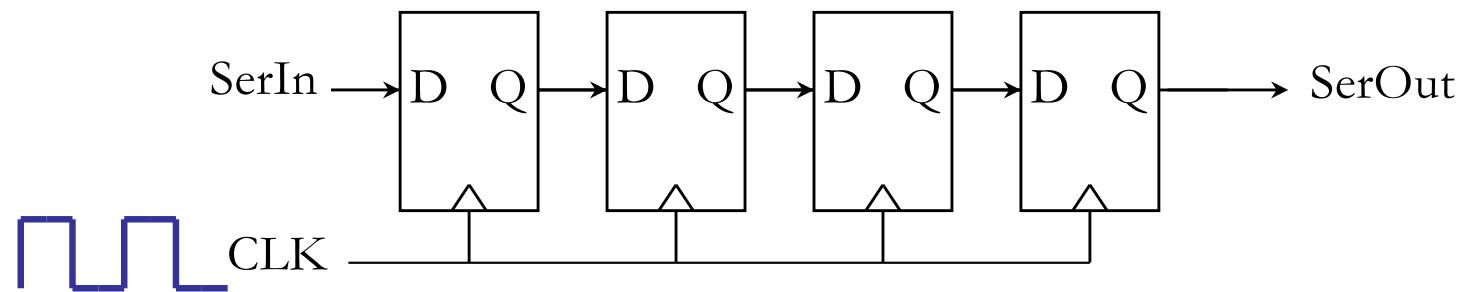
Also called a *parallel-in/parallel-out register* because the flip-flops operate in parallel

A Shift Register



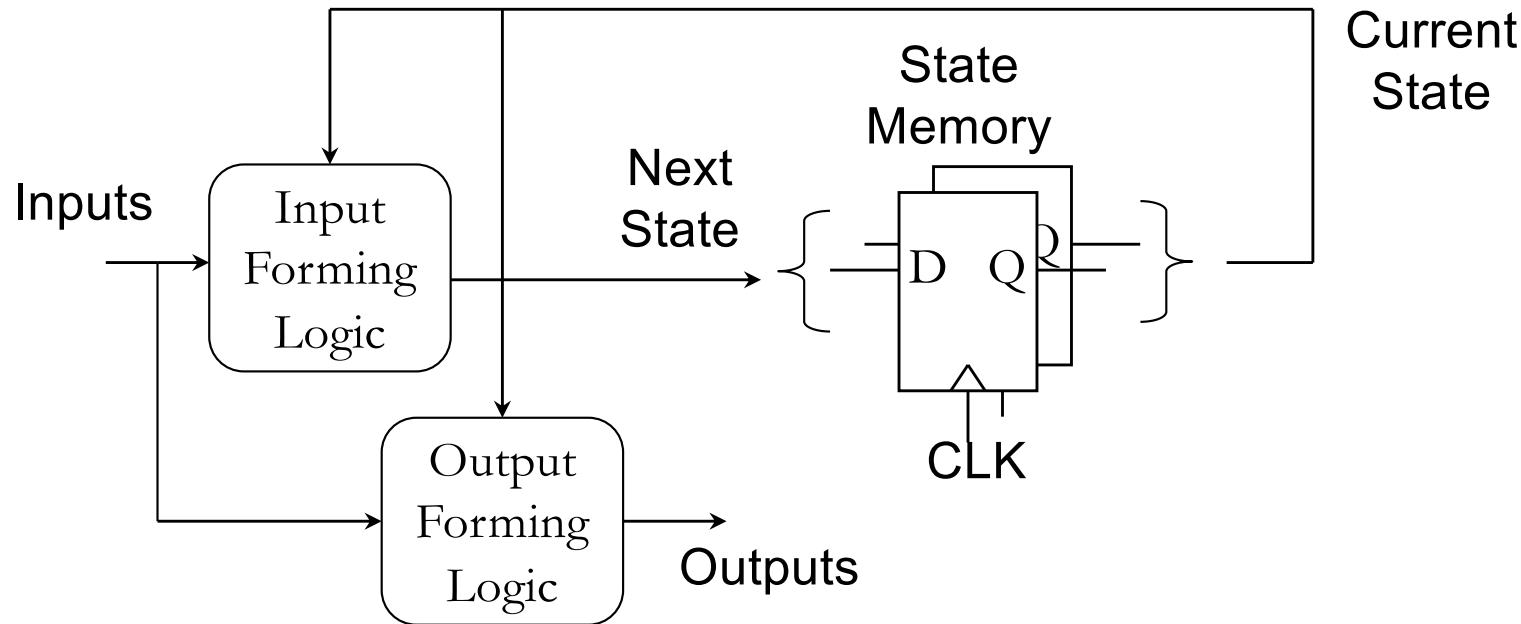
Called a *serial-in, parallel-out shift register*

Serial-In/Serial-Out Shift Register



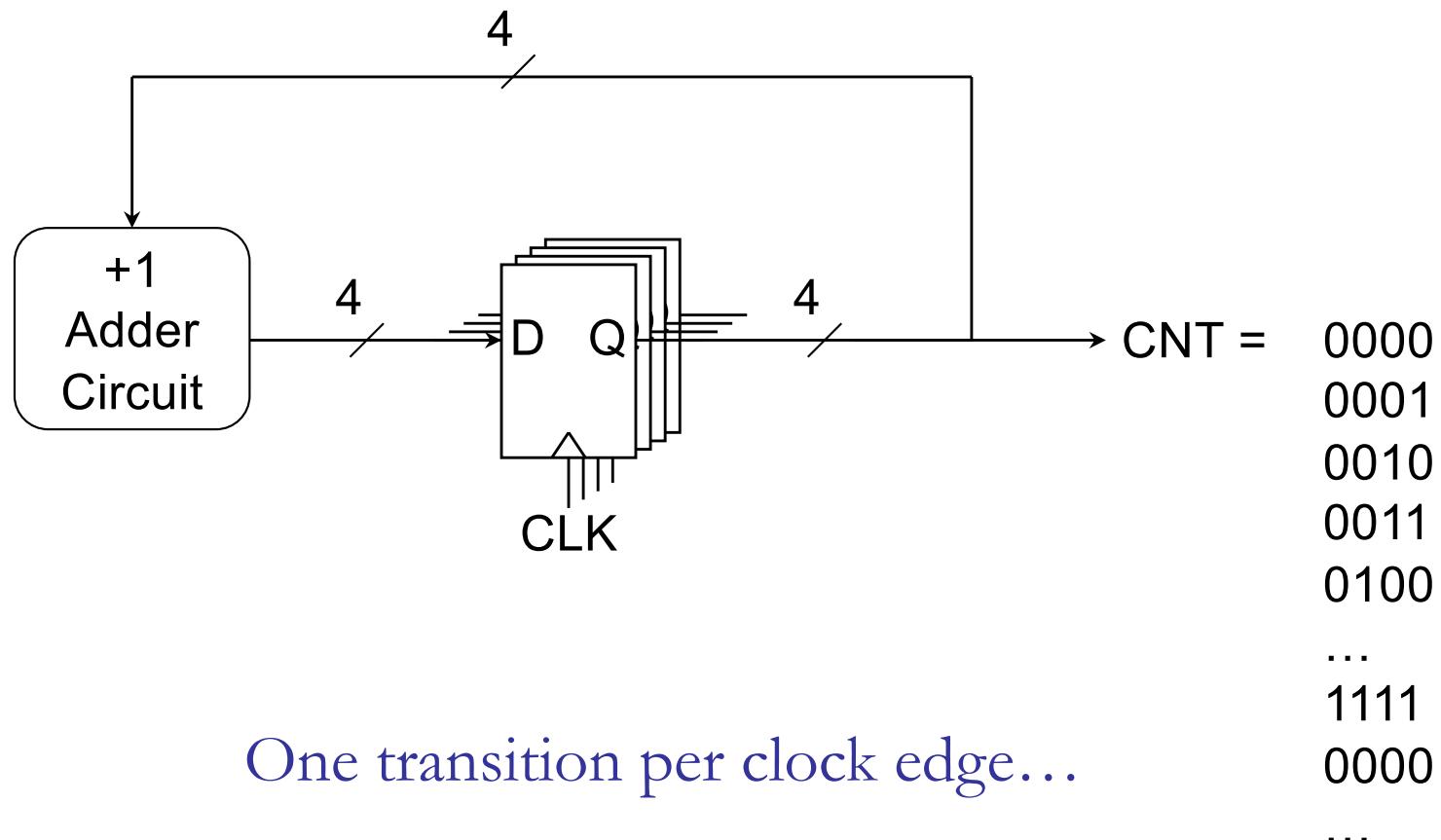
Useful for delaying a serial bit-stream some number of cycles...

General Sequential System

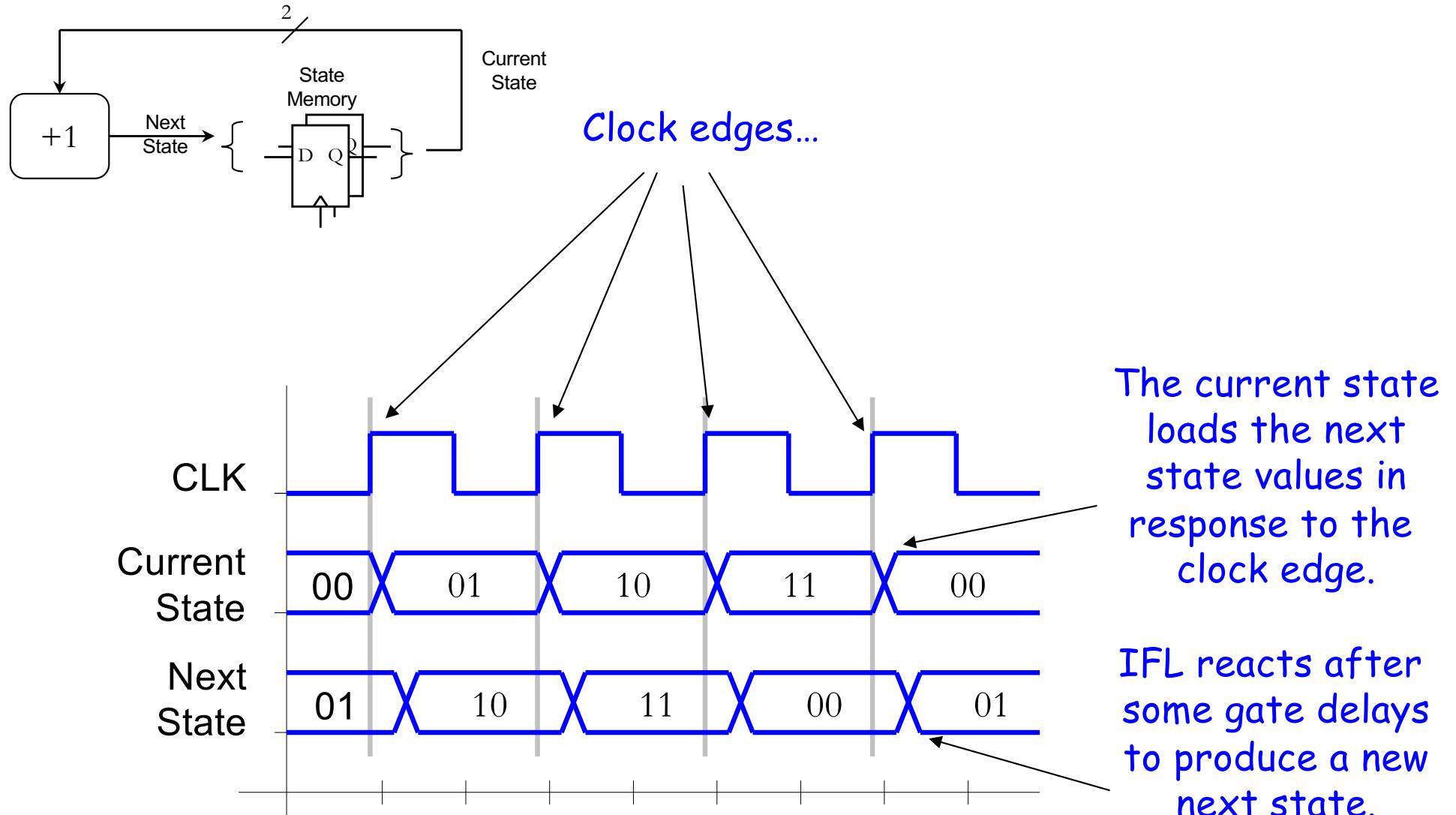


- Globally Synchronous: One global clock
- All registers load on that clock's edge
- Input forming logic determines what is loaded
- Simplifies timing analysis and requirements

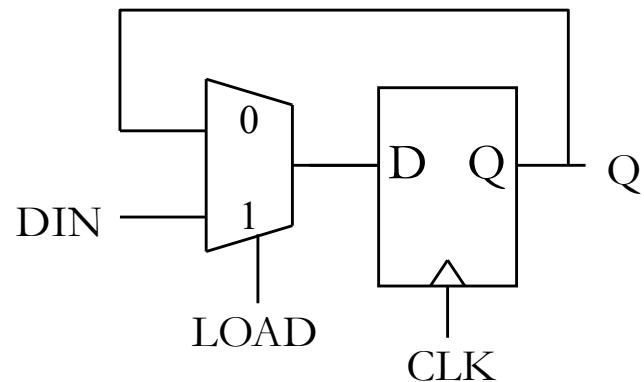
Example of a Synchronous System: A Sequential Counter



A Sequential Counter



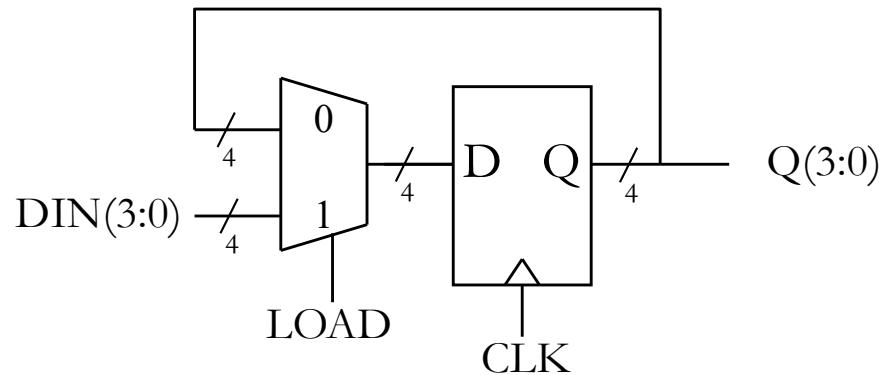
A Loadable Register (1-Bit)



When LOAD=0, Flip-Flop loads old value

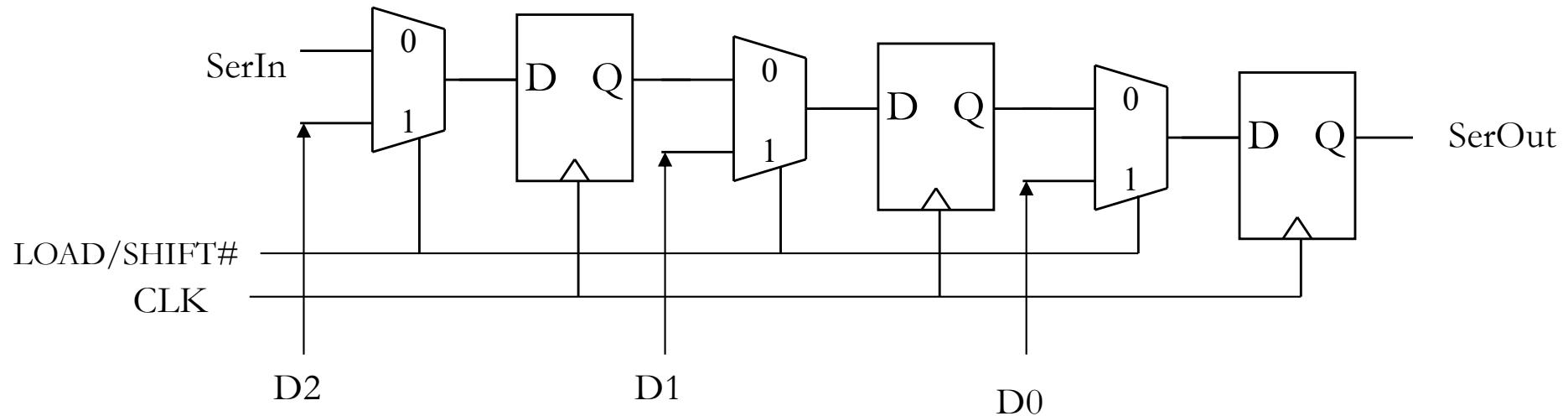
When LOAD=1, Flip-Flop loads DIN

A Loadable Parallel-In, Parallel-Out Register



DIN gets stored in register on rising edge of CLK
but only when LOAD = 1

A Loadable Parallel-In, Serial-Out Register

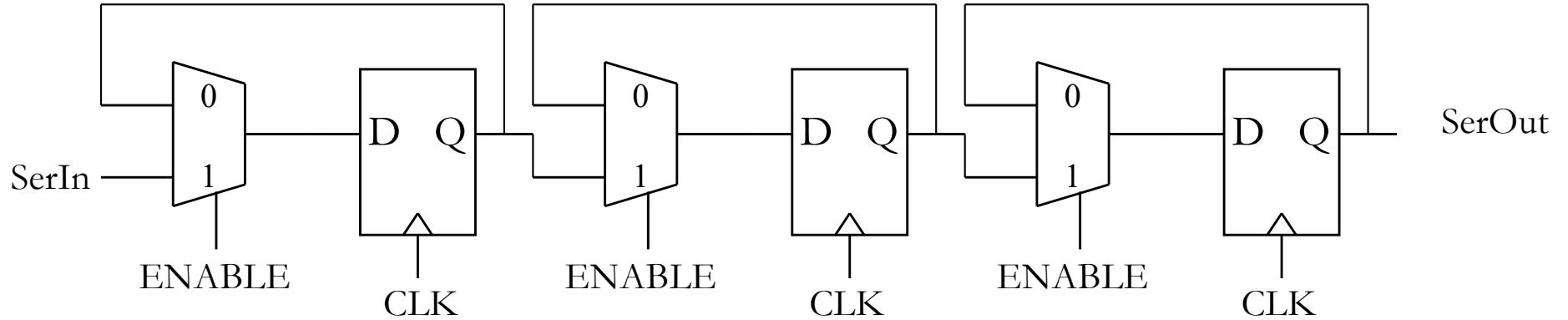


When LOAD/SHIFT# = 1, register loads D2...D0

When LOAD/SHIFT# = 0, register shifts right

This register is always either loading or shifting

A Serial-In Serial-Out Register With An Enable Input

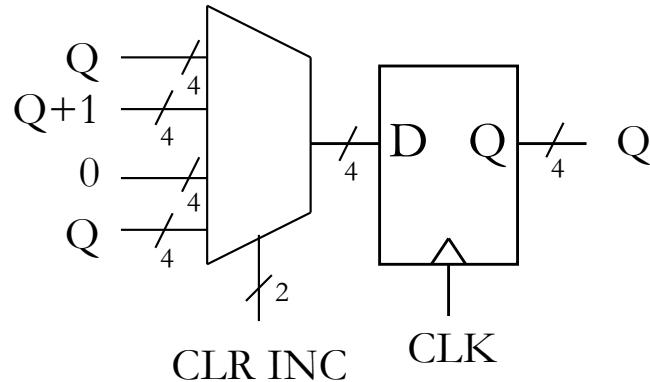


Combination of loadable register with shift register...

When ENABLE=0, register doesn't shift (re-loads old value)

When ENABLE=1, register shifts

A Clearable Counter

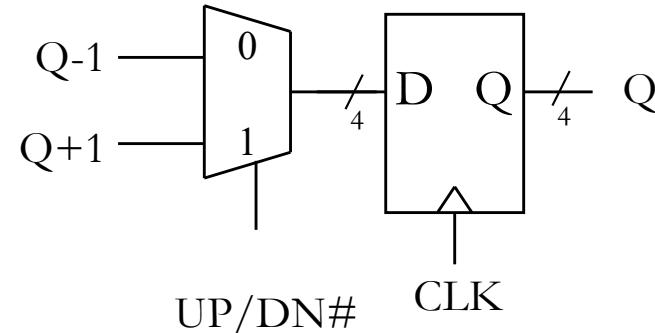


CLR	INC	Q+
0	0	Q
0	1	Q+1
1	0	0
1	1	Q

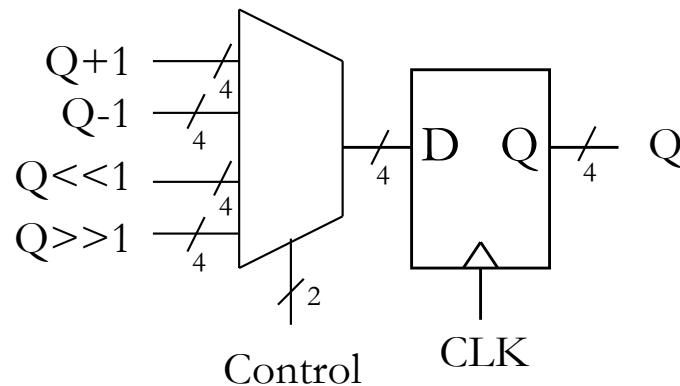
- The loadable register concept can be generalized to provide any combination of inputs to register

Load, clear, set, enable, left shift, right shift, increment, decrement, etc.

An Up/Down Counter



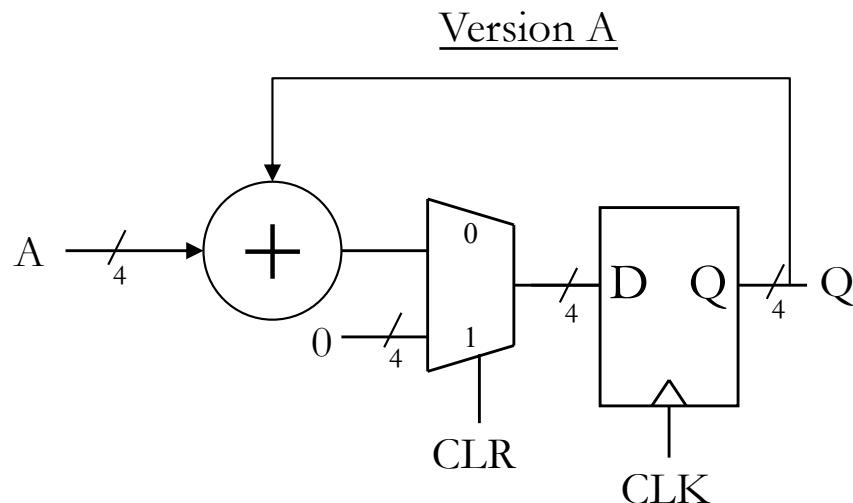
Up/Down Counter with Bi-Directional Shift Register



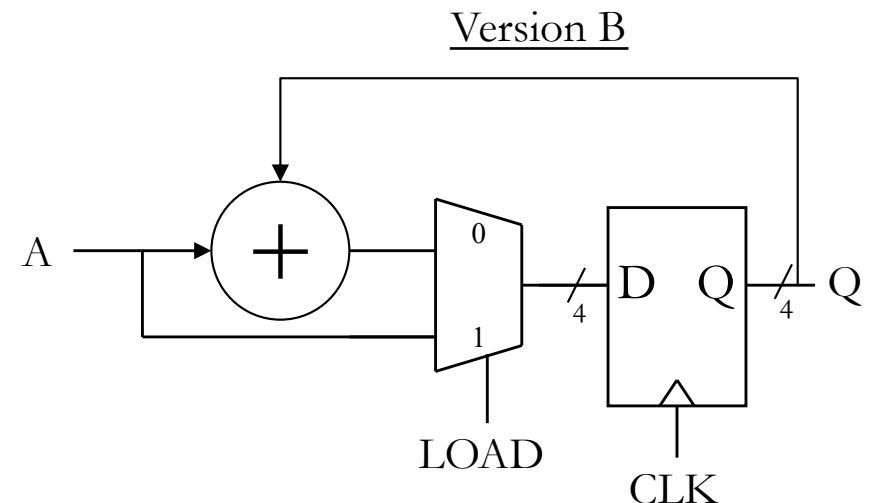
Control	NextQ
00	$Q+1$
01	$Q-1$
10	Q shifted left
11	Q shifted right

An Accumulator

- Values to be added are placed on A input, one per cycle. Register accumulates their sum.



This one loads 0 when CLR=1

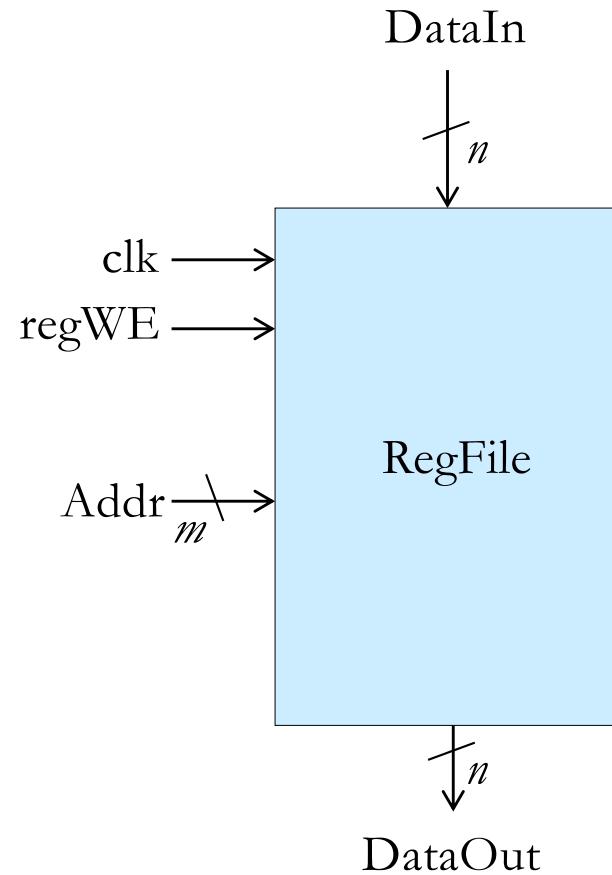


This one loads A when LOAD=1

Register Files

Small memories holding multiple words of
data

Typical Register File



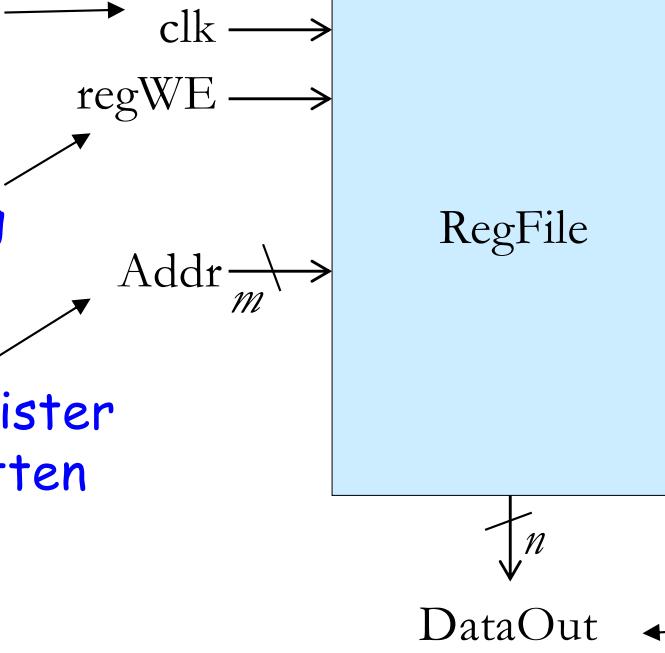
Typical Register File

Reads are *asynchronous* (combinational)

Writes occur on the clock edge.

Controls whether reading or writing

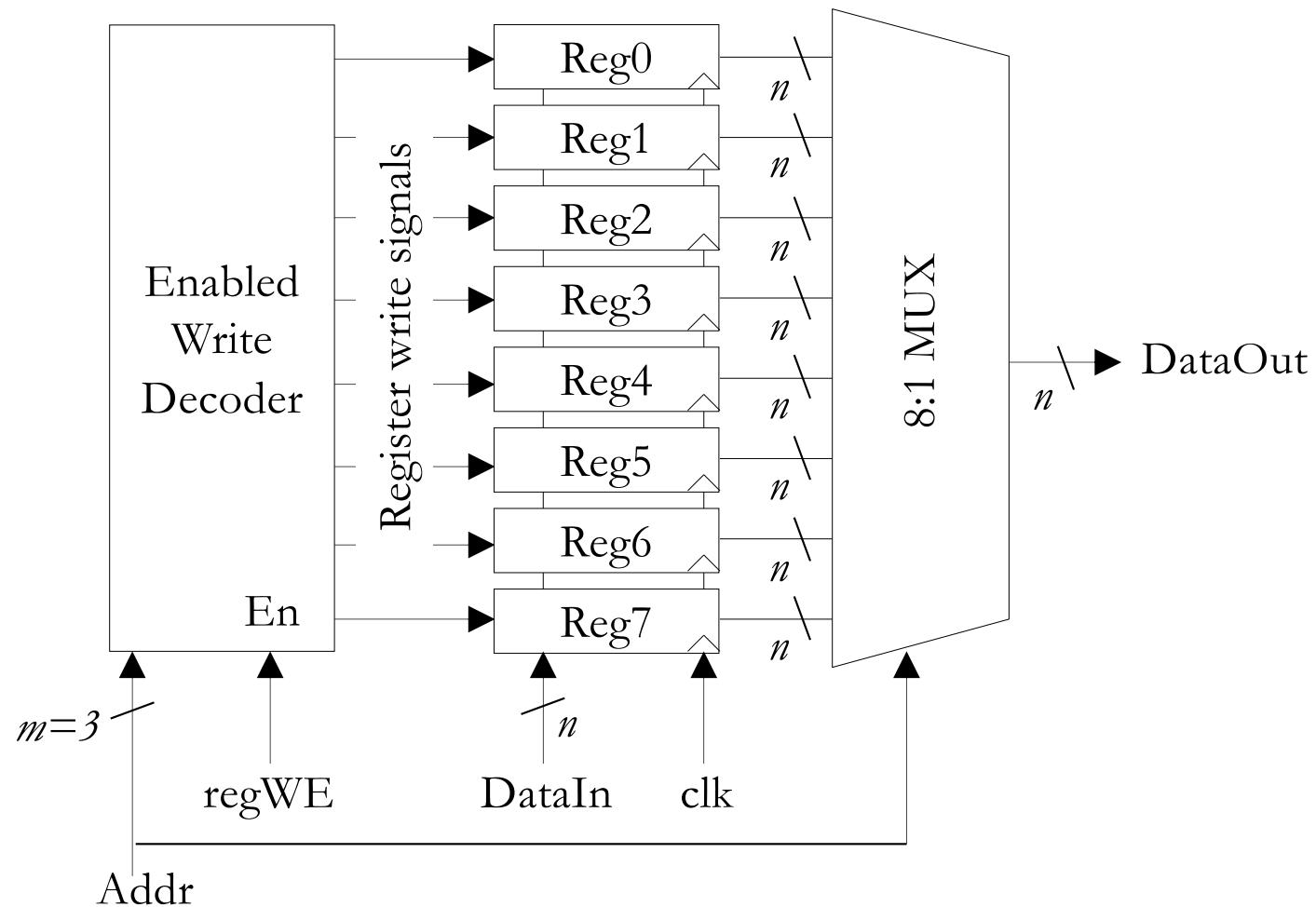
Address in the register to be read or written



Data to be written to register file

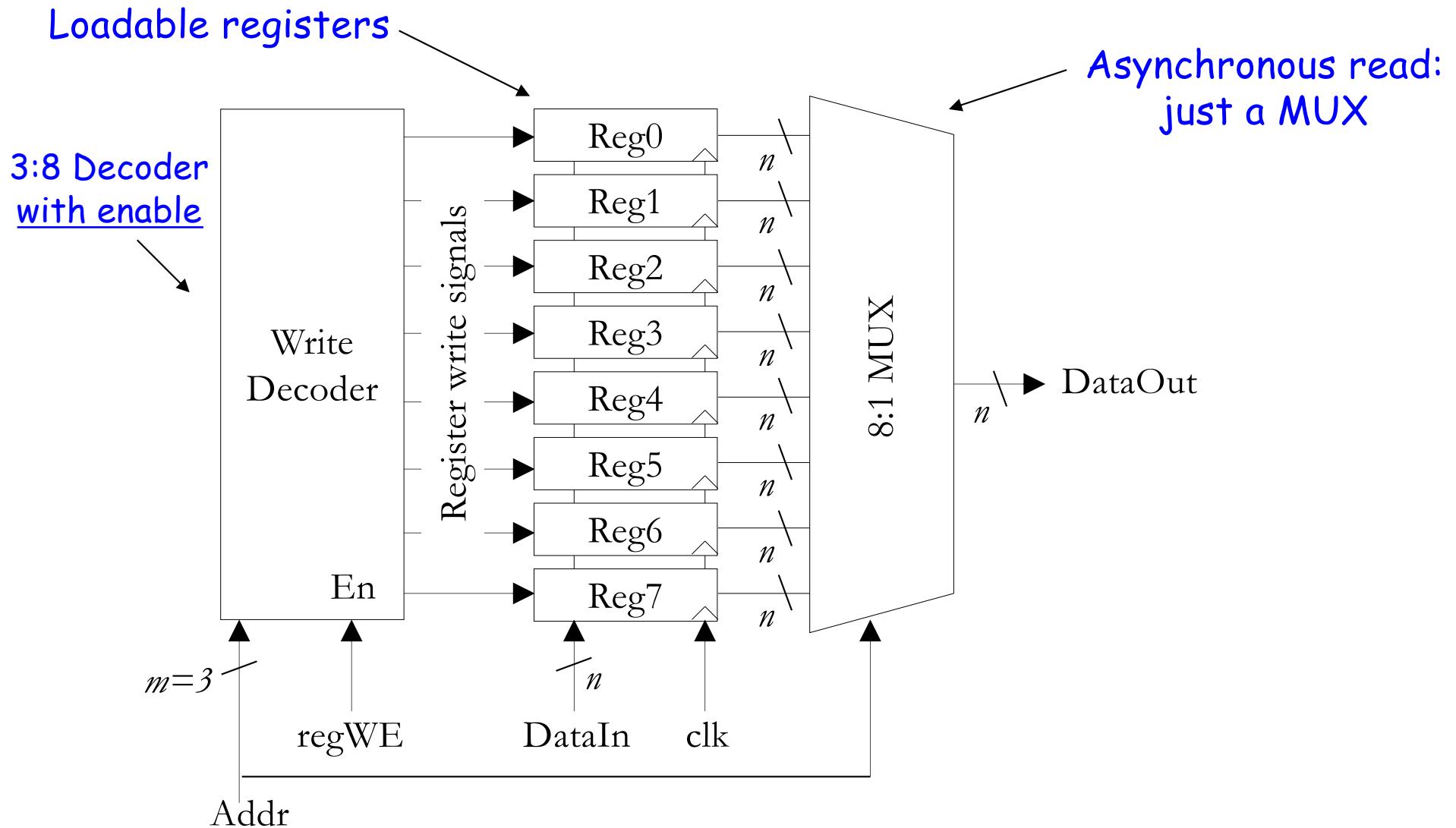
This register file will hold 2^m registers
Each register is n -bits wide

Building a Register File



An enabled decoder acts like a decoder when enable = 1.
Outputs all zeros when enable = 0.

Building a Register File



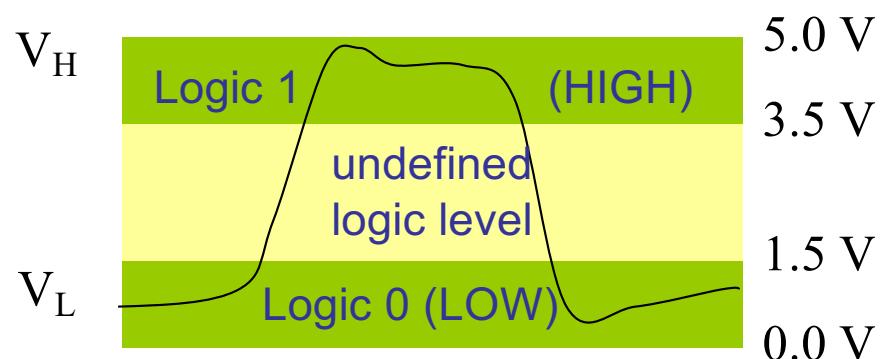
CMOS Digital Circuits

The building blocks of digital circuits

Voltage Defined Logic

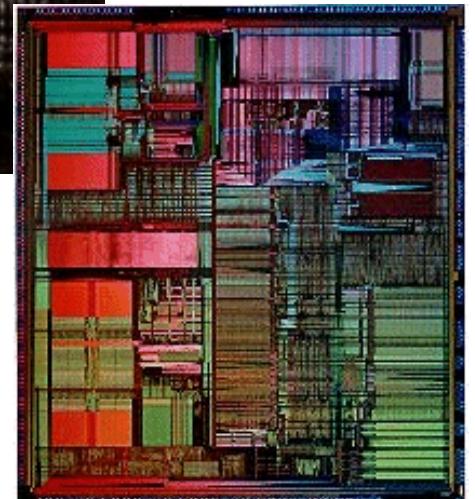
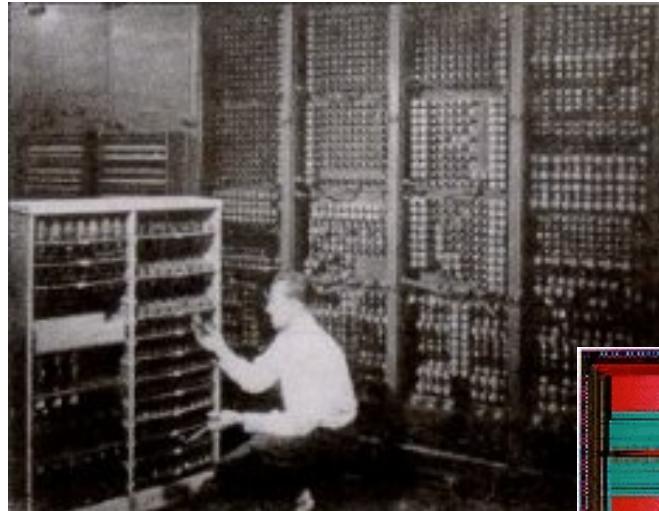
- Digital signals are actually analog!
- Logic level specified by a voltage range
 - Low region
 - High region
 - Undefined region
- Voltage levels decreasing with advanced processes

Example: 5V CMOS



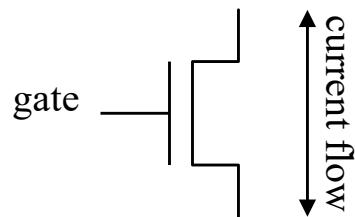
Logic Families

- 1940's Relay logic
- 1950's Vacuum Tubes
- 1960's TTL (bipolar transistors)
- 1970's MOS
- 1980's CMOS
- 1990's CMOS
- 2000's CMOS
- 2010's CMOS
- 2020's Probably CMOS

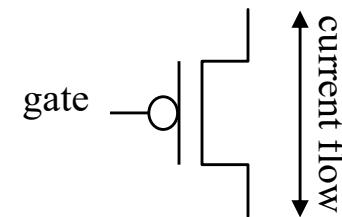


The MOS Transistor

- Acts like a switch
 - conducts current when in the “on” state



Off = open circuit
On = closed circuit

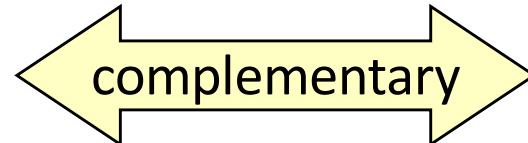


N-type Transistor

gate FET	
0	off
1	on

P-type Transistor

gate FET	
0	on
1	off

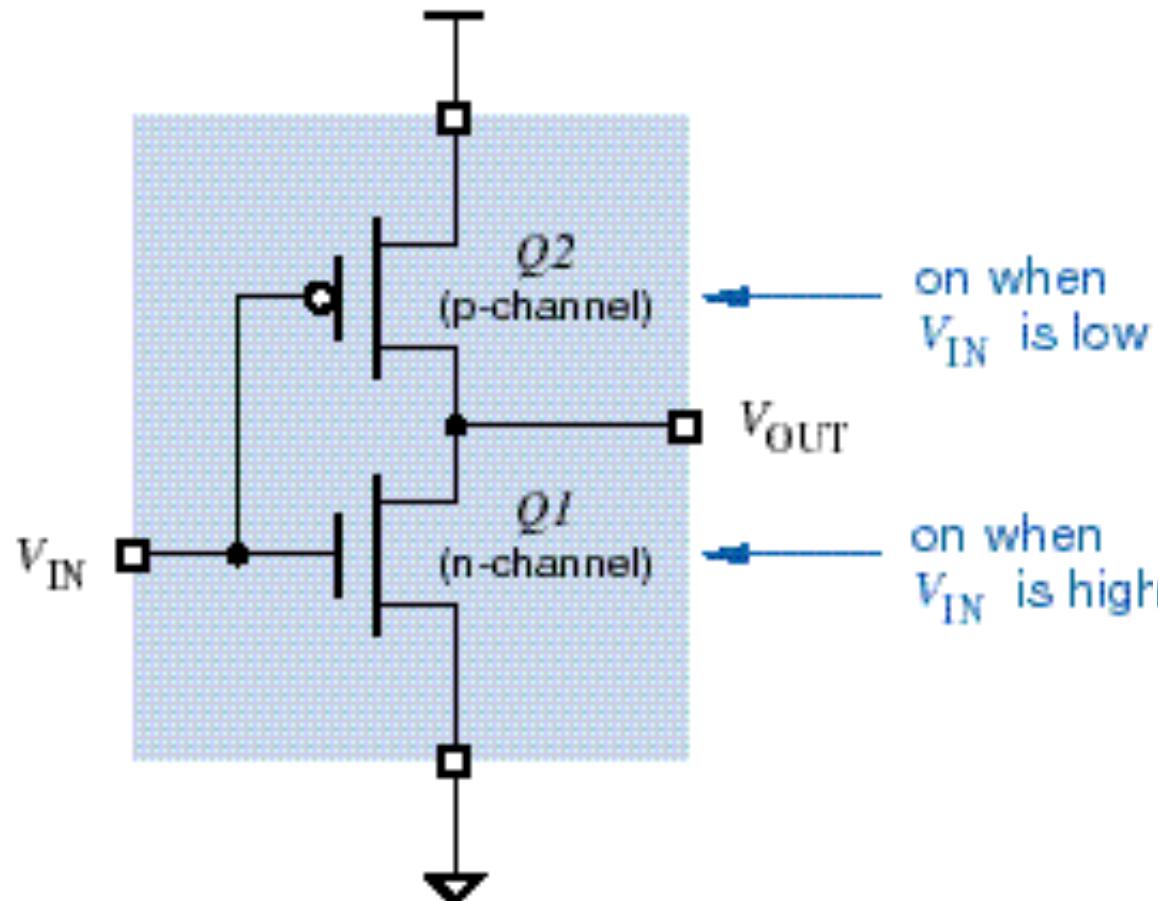


MOS = metal-oxide semiconductor

CMOS = complementary MOS with both N and P transistors

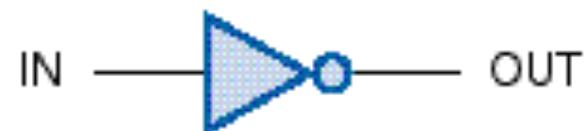
CMOS Inverter

$V_{DD} = +5.0 \text{ V}$

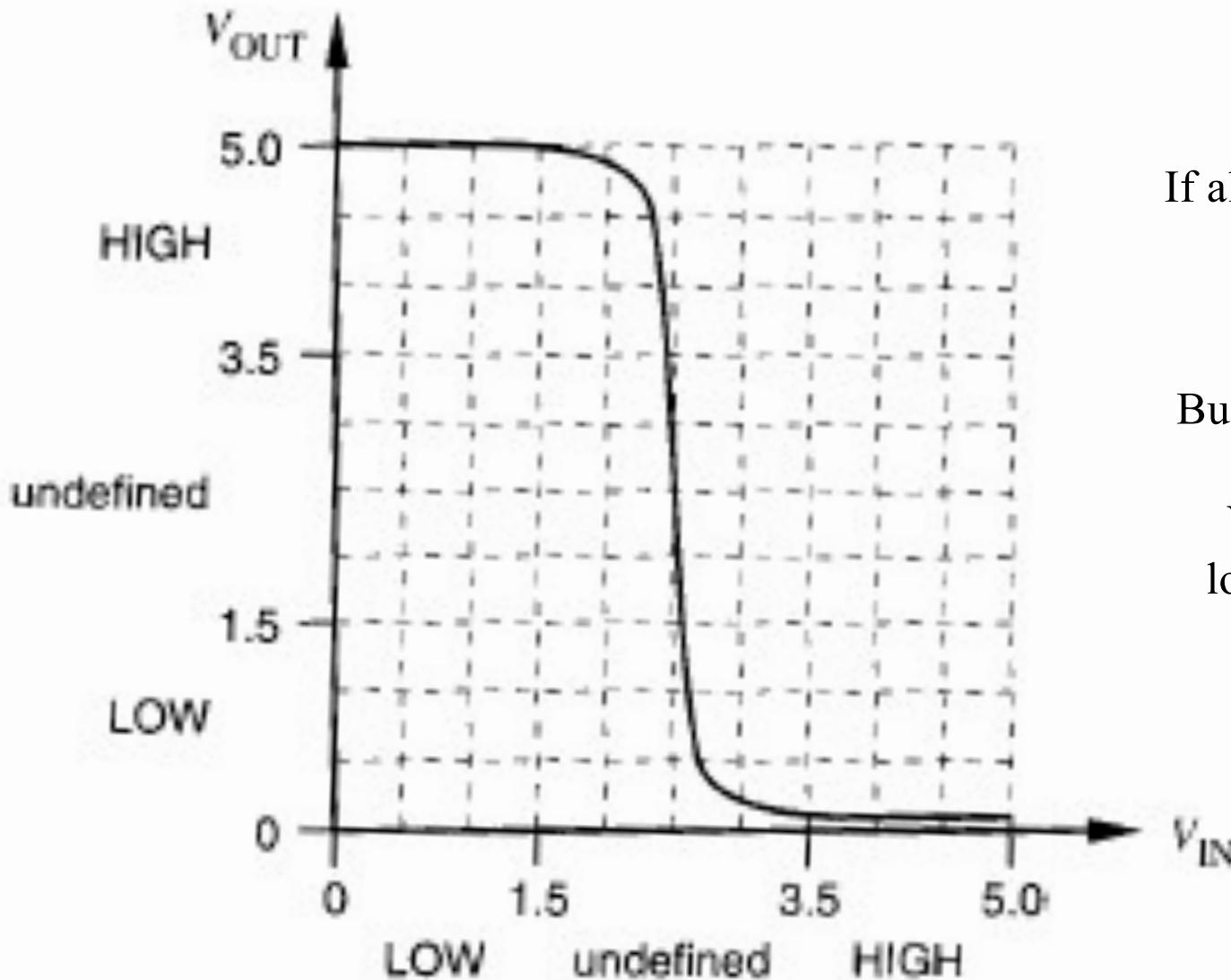


Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

V_{IN}	$Q1$	$Q2$	V_{OUT}
0.0 (L)	off	on	5.0 (H)
5.0 (H)	on	off	0.0 (L)



CMOS input-output transfer curve



If all inverters were like this:

$$'0' < 2.4V$$

$$'1' > 2.6V$$

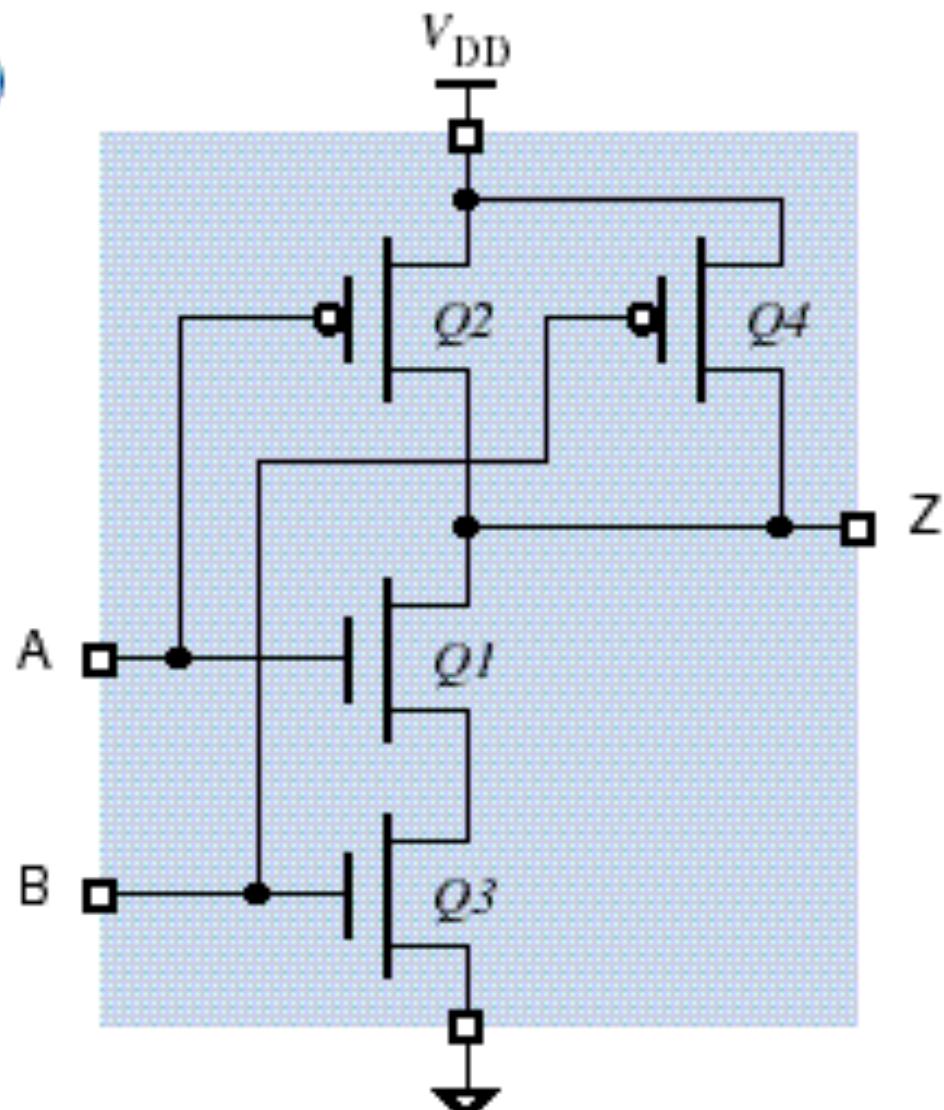
But this is merely *typical*...

Varies based on: V_{cc} ,
loading, temperature, ...

CMOS NAND Gate

- Use $2n$ transistors for n -input gate

(a)

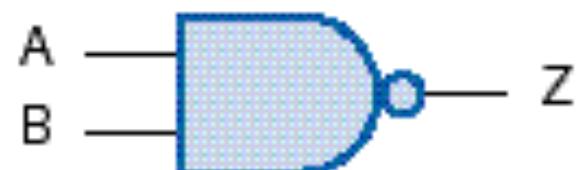


Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

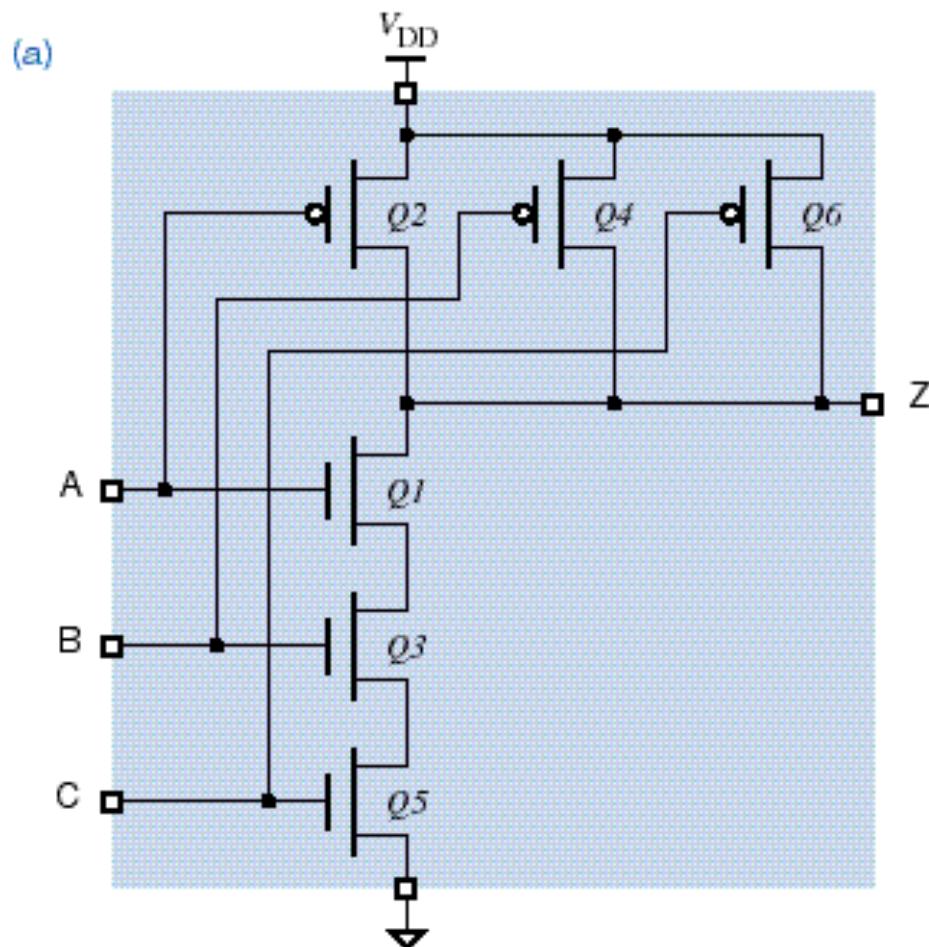
(b)

A	B	$Q1$	$Q2$	$Q3$	$Q4$	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	H
H	L	on	off	off	on	H
H	H	on	off	on	off	L

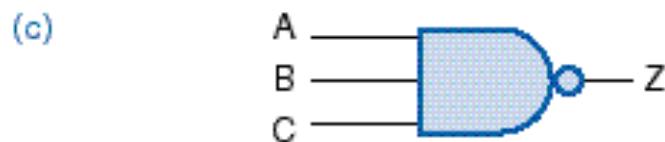
(c)



CMOS NAND Gate – 3 inputs

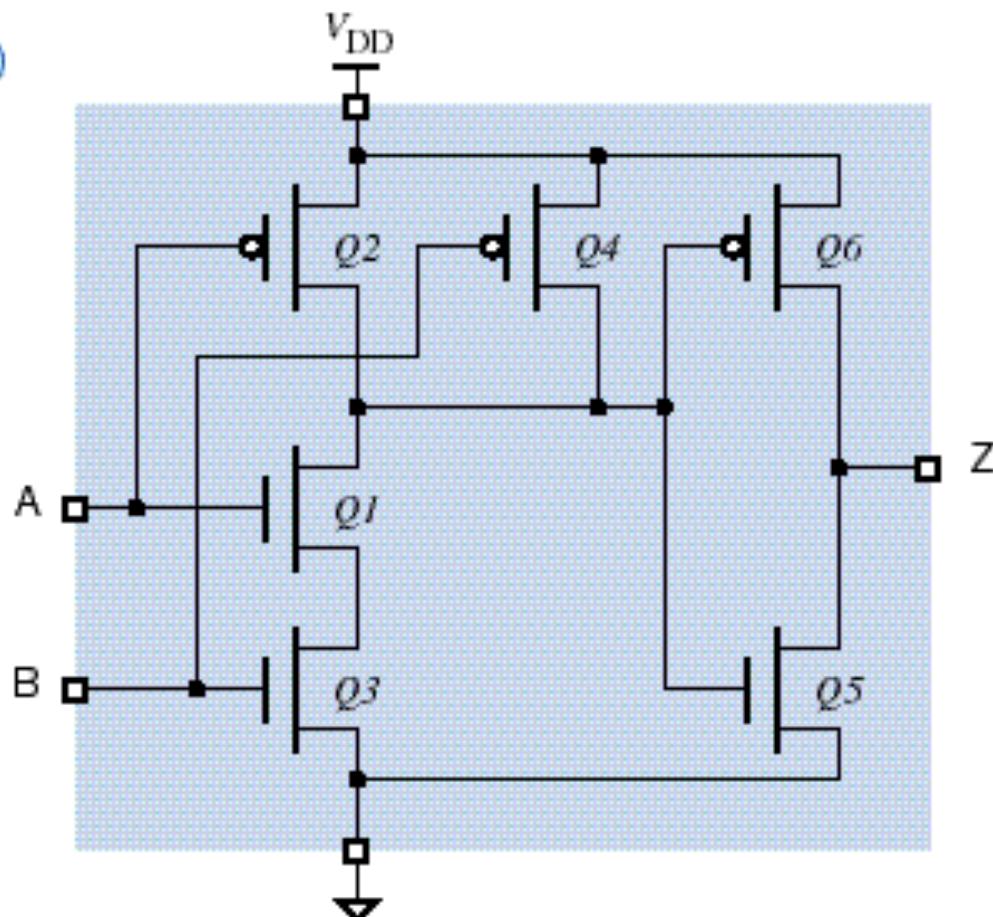


A	B	C	$Q1$	$Q2$	$Q3$	$Q4$	$Q5$	$Q6$	Z
L	L	L	off	on	off	on	off	on	H
L	L	H	off	on	off	on	on	off	H
L	H	L	off	on	on	off	off	on	H
L	H	H	off	on	on	off	on	off	H
H	L	L	on	off	off	on	off	on	H
H	L	H	on	off	off	on	on	off	H
H	H	L	on	off	on	off	off	on	H
H	H	H	on	off	on	off	on	off	L



2-input AND gate

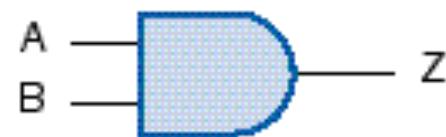
(a)



(b)

A	B	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	off	on	off	on	on	off	L
L	H	off	on	on	off	on	off	L
H	L	on	off	off	on	on	off	L
H	H	on	off	on	off	off	on	H

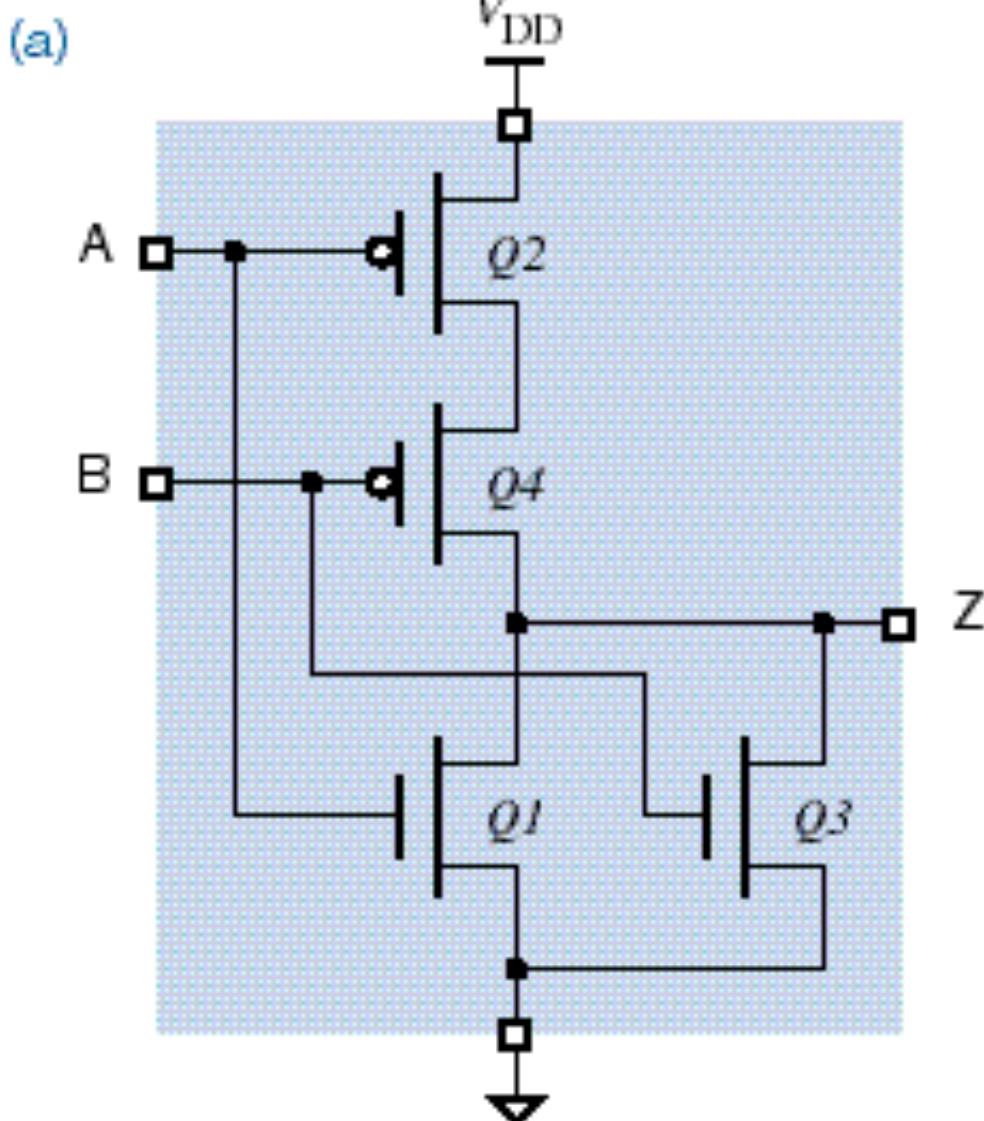
(c)



Copyright © 2000 by Prentice Hall, Inc.
Digital Design Principles and Practices, 3/e

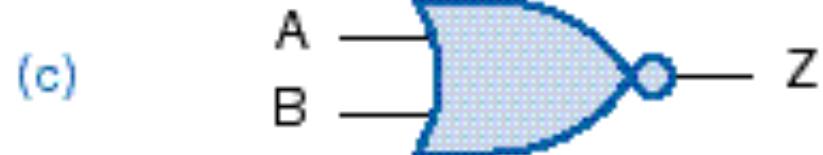
CMOS NOR Gate

- Like NAND -- $2n$ transistors for n -input gate



(b)

A	B	$Q1$	$Q2$	$Q3$	$Q4$	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	L
H	L	on	off	off	on	L
H	H	on	off	on	off	L

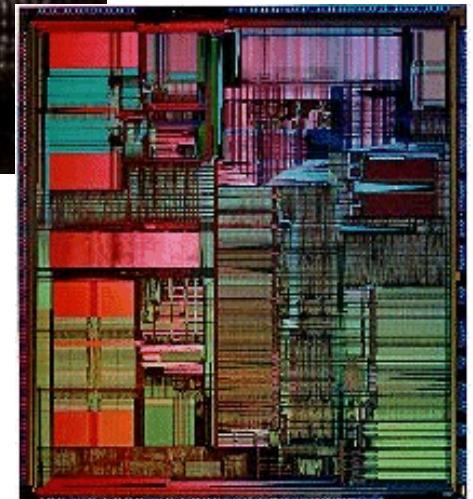
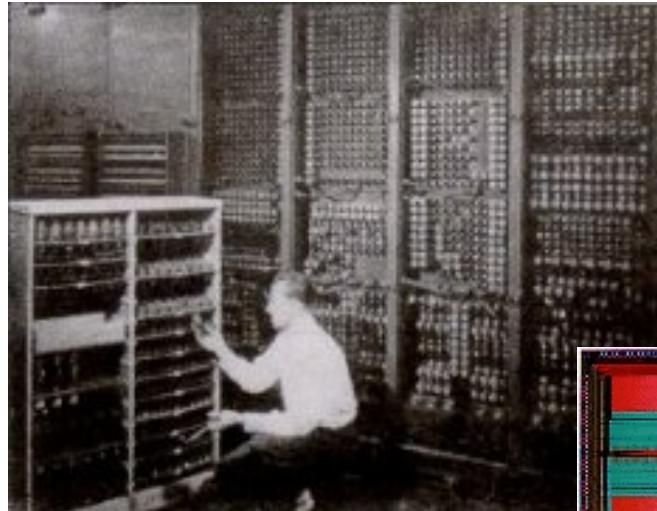


Lecture 22

Types of Memory
Microcontrollers
Arduino Platform

Technology for Building Computers

- 1940's Relay logic
- 1950's Vacuum Tubes
- 1960's TTL (bipolar transistors)
- 1970's MOS
- 1980's CMOS
- 1990's CMOS
- 2000's CMOS
- 2010's CMOS
- 2020's Probably CMOS

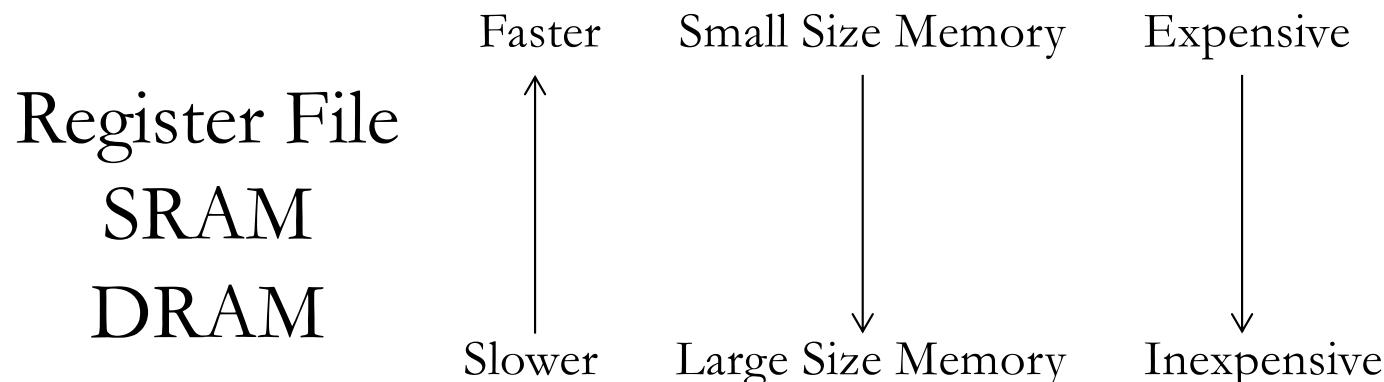


A Register File vs. RAM

- Computer circuits all depend upon the presence of memory for both data and programs.
 - A flip-flop can store one bit of information.
 - A register can store a single “word,” typically 8-64 bits.
 - A register file can store data in a small array of addressed memory.
 - RAM can store data in a very large array of addressed memory.
- Random Access Memory (RAM) is conceptually similar to a register file (it stores data in addressed memory), but RAM uses different technology and is much, much larger (Gbytes instead of bytes), slower and less expensive.

Random Access Memory (RAM)

- Random access memory, or RAM, allows us to store large amounts of data.
 - You can write a value and then read the value that was saved.
 - An address specifies which memory value to read or write.
 - Each value can be a multiple-bit data word (like 8-bits).
- There are different technologies of RAM, such as SRAM (Static RAM) and DRAM (Dynamic RAM)



Memory Model

- You can think of computer memory as being one big array of data.
 - The address serves as an array index.
 - Each address points to one word of memory data.
- You can read or write the data at any given memory address

Store 9 to address 2.

Store 25 to address 4.

Store 15 to address 6.

Load data from address 4.

What data is read from memory by the load instruction?

Address	Data
00000000	
00000001	
00000002	
.	
.	
.	
.	
.	
.	
.	
.	
.	
.	
.	
.	
FFFFFFFFFFD	
FFFFFFFFFFE	
FFFFFFFFFFF	

For memory with k address bits and n bytes of data per address, the total size of memory is $= n2^k$

Types of Memory

- Non-volatile Memory
 - Retains memory when powered-off.
 - Slow to write, fast to read.
 - Flash memory (like a thumb drive)
 - Electrically Erasable Programmable Read Only Memory (EEPROM), older technology but still in use
- Volatile
 - Static RAM (**SRAM**), fast but costs more than DRAM.
 - Dynamic RAM (**DRAM**), slower but cheap. You can afford lots of it.

A Microcontroller (MCU) is ...

- Low cost, low power computer system on a chip
 - a **processor core**, with...
 - on-chip **flash memory** (non-volatile) for storing programs
 - on-chip **static RAM** (SRAM) for storing data
 - built in programmable input/output **peripherals**
- Designed for “embedded systems” applications
 - Dedicated computer systems embedded inside the devices they control
 - Different than microprocessors which are used in personal computers or other general purpose applications

Common Microcontroller Peripherals

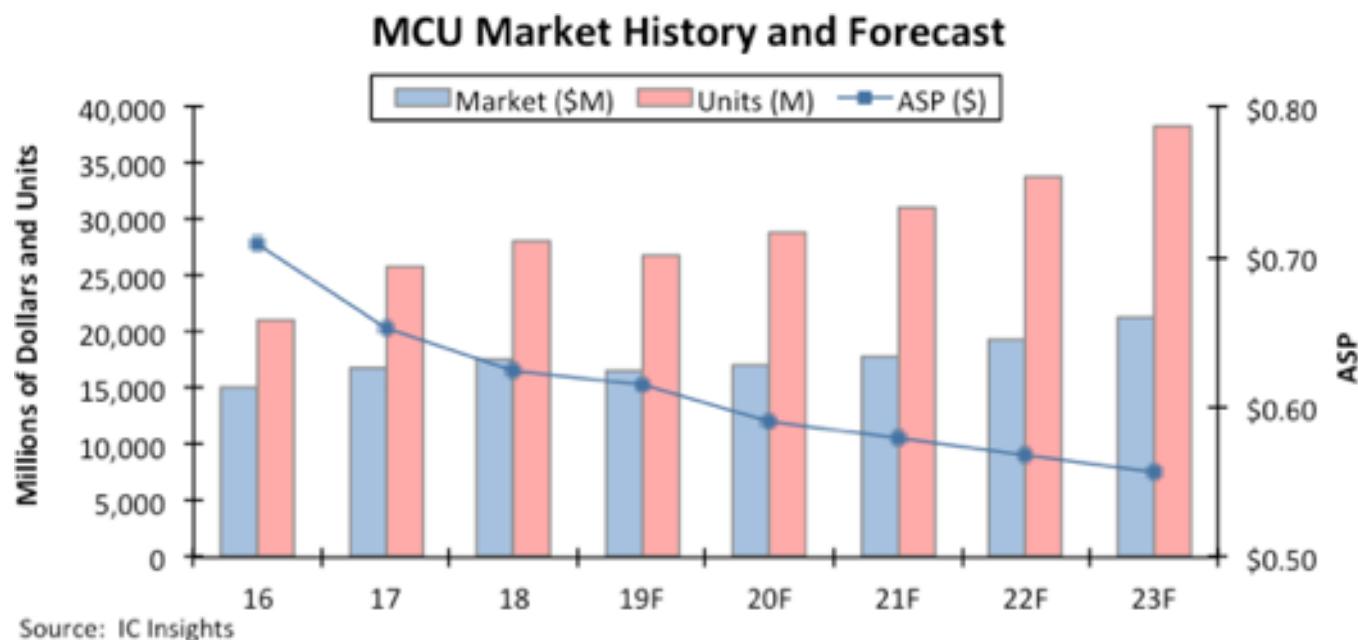
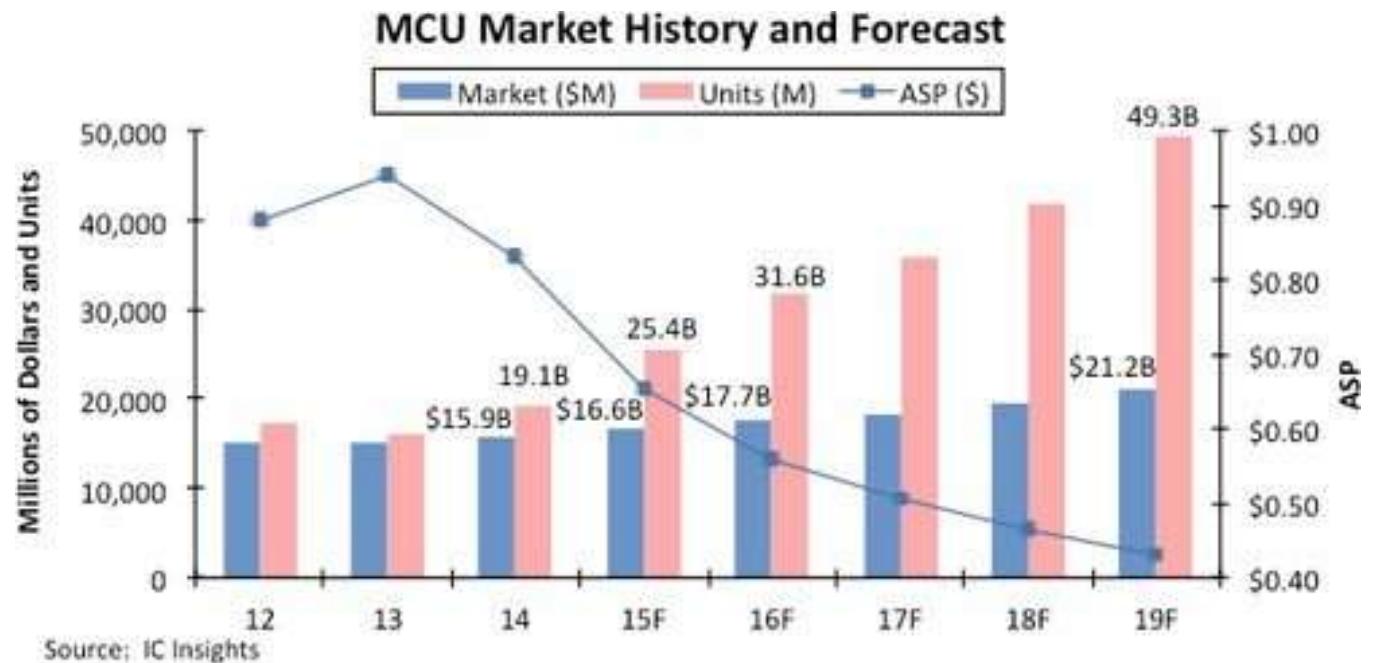
- General purpose input/output pins (GPIO)
 - Software configurable to either an input or an output state
 - Inputs are used to read sensors or external signals.
 - Outputs drive external devices such as LEDs or motors.
- Analog-to-digital converter (ADC)
 - Samples analog signals and converts them into digital signals.
- Digital-to-analog converter (DAC)
 - Converts digital numbers to analog voltages or currents.
- Pulse Width Modulation (PWM)
 - Generates an oscillating signal on a GPIO pin at a well defined frequency/duty cycle

Common Microcontroller Peripherals

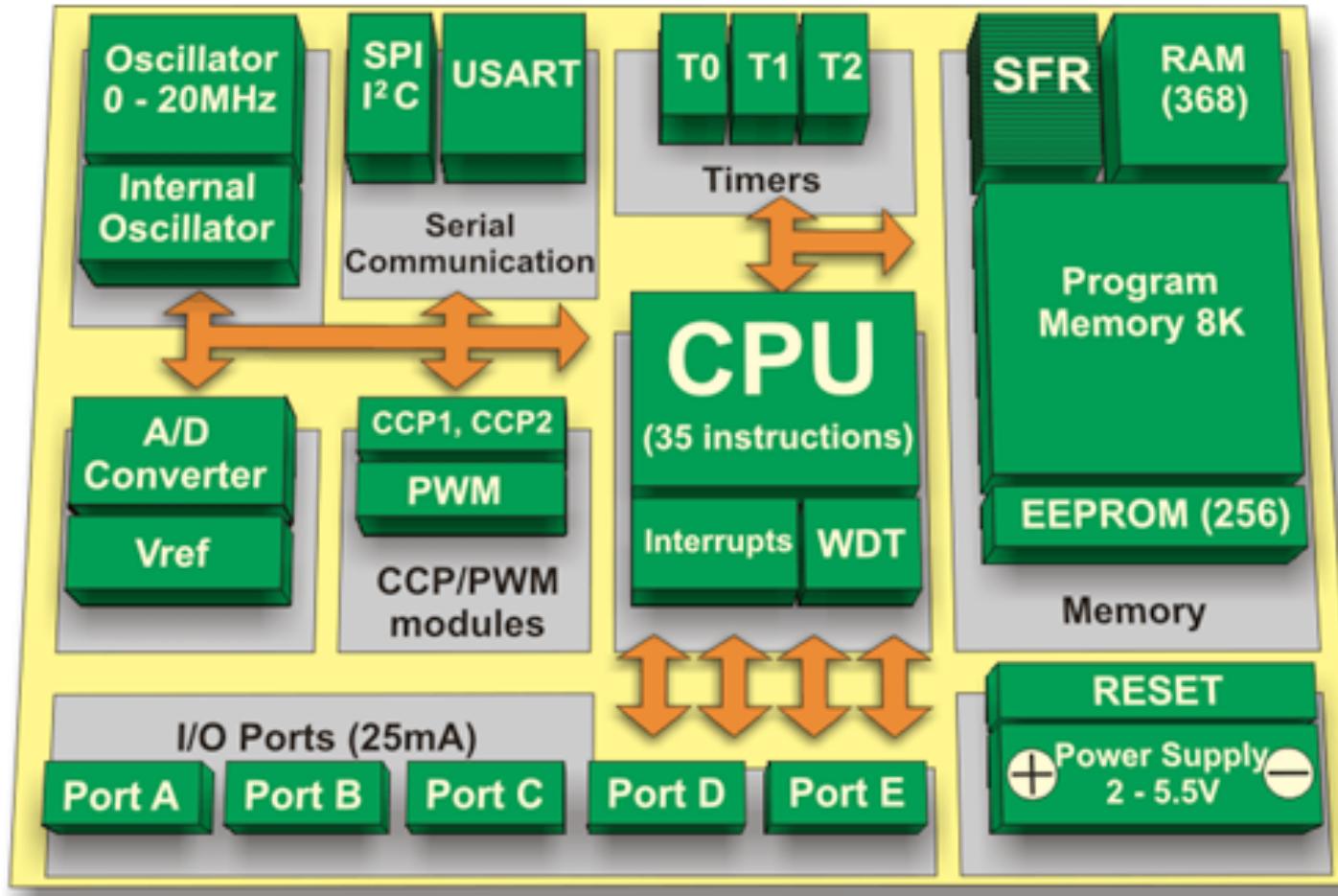
- Programmable Timers
 - generate periodic interrupts or time internal/external events
- Universal Asynchronous Receiver/Transmitter (UART)
 - receiving and transmitting data over a serial line with very little load on the CPU (such as RS232/RS485)
- Serial Peripheral Interface (SPI)
 - Synchronous serial, master-slave, full-duplex, 4 wire interface for connecting to sensors and actuators.
- Inter-Integrated Circuit (I²C)
 - Synchronous serial, master-slave, half-duplex, 2 wire interface for connecting to sensors and actuators.

Internal Hardware Units

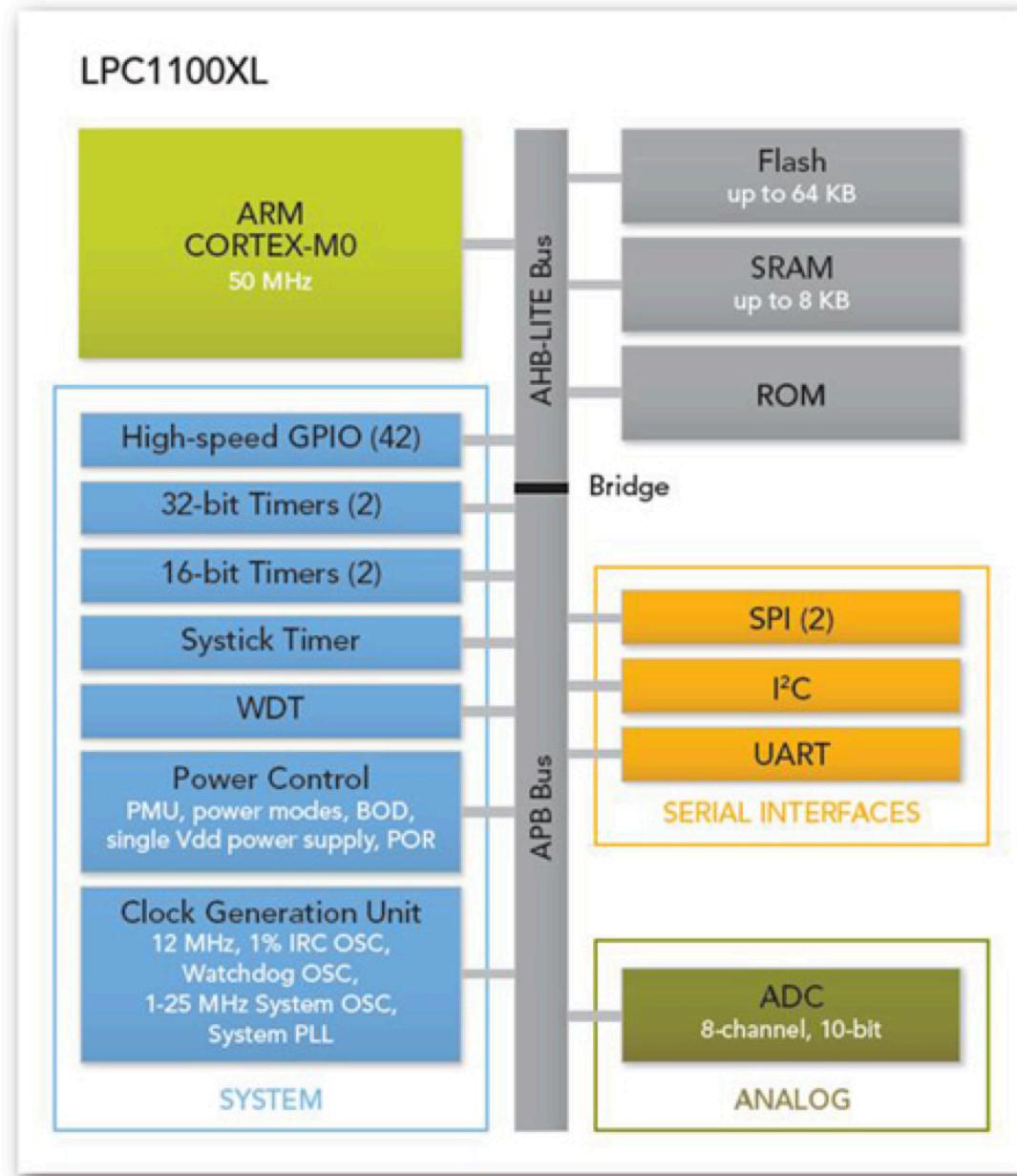
- Watchdog Timer
 - A timer called the watch dog. You must “pet the dog” regularly or it will “bark” and re-boot the microcontroller.
- EEPROM
 - Non-volatile memory (does not get erased when power goes out) for data. Mostly for configuration parameters.
- Power Management – Save power when processor is not being used.
- Power-on Reset – put processor in known state when it is powered on.
- Brown-out Detection – detects sagging supply voltage
- Clock Management – circuitry to generate the clock



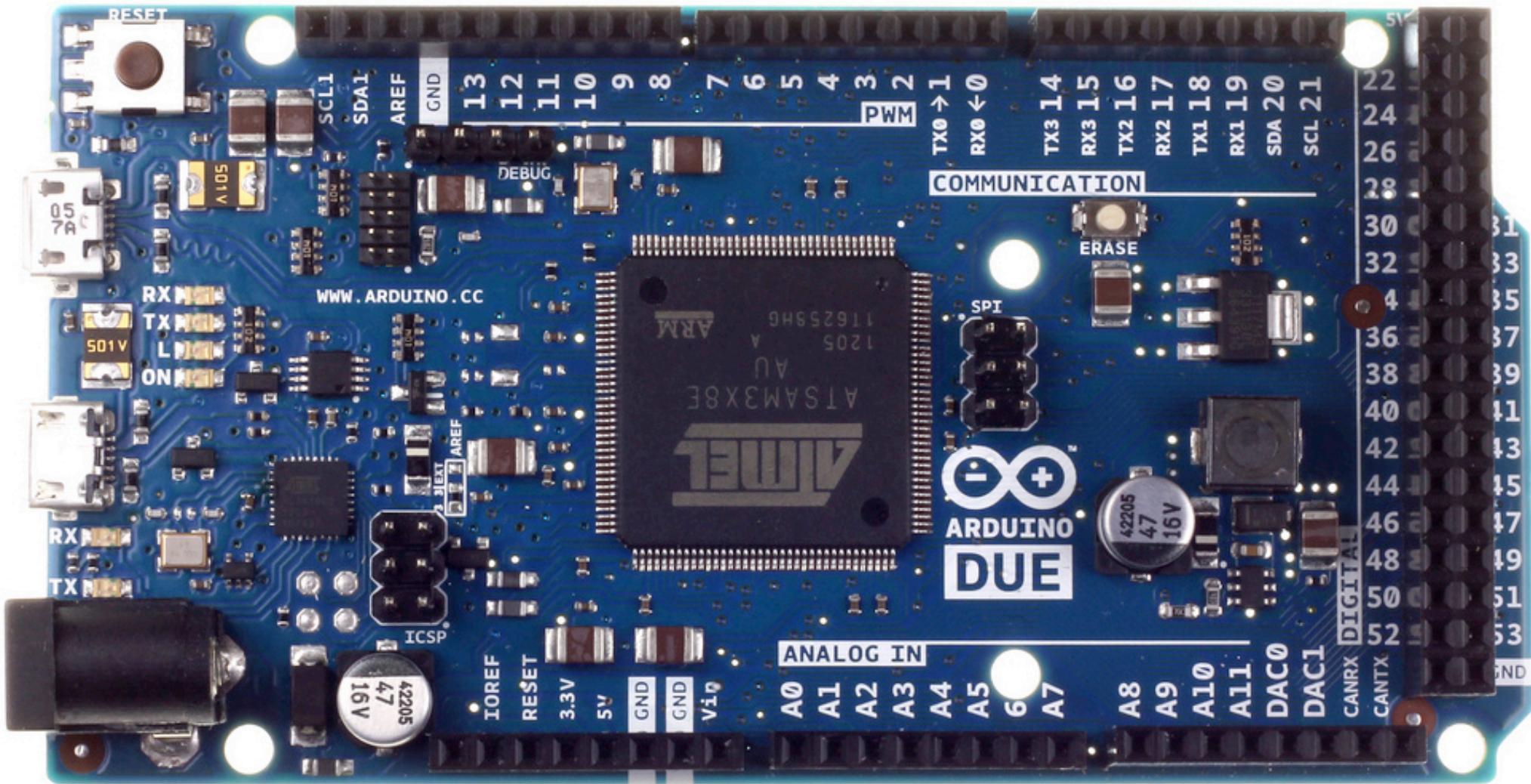
MicroChip PIC Architecture



NXP ARM Architecture

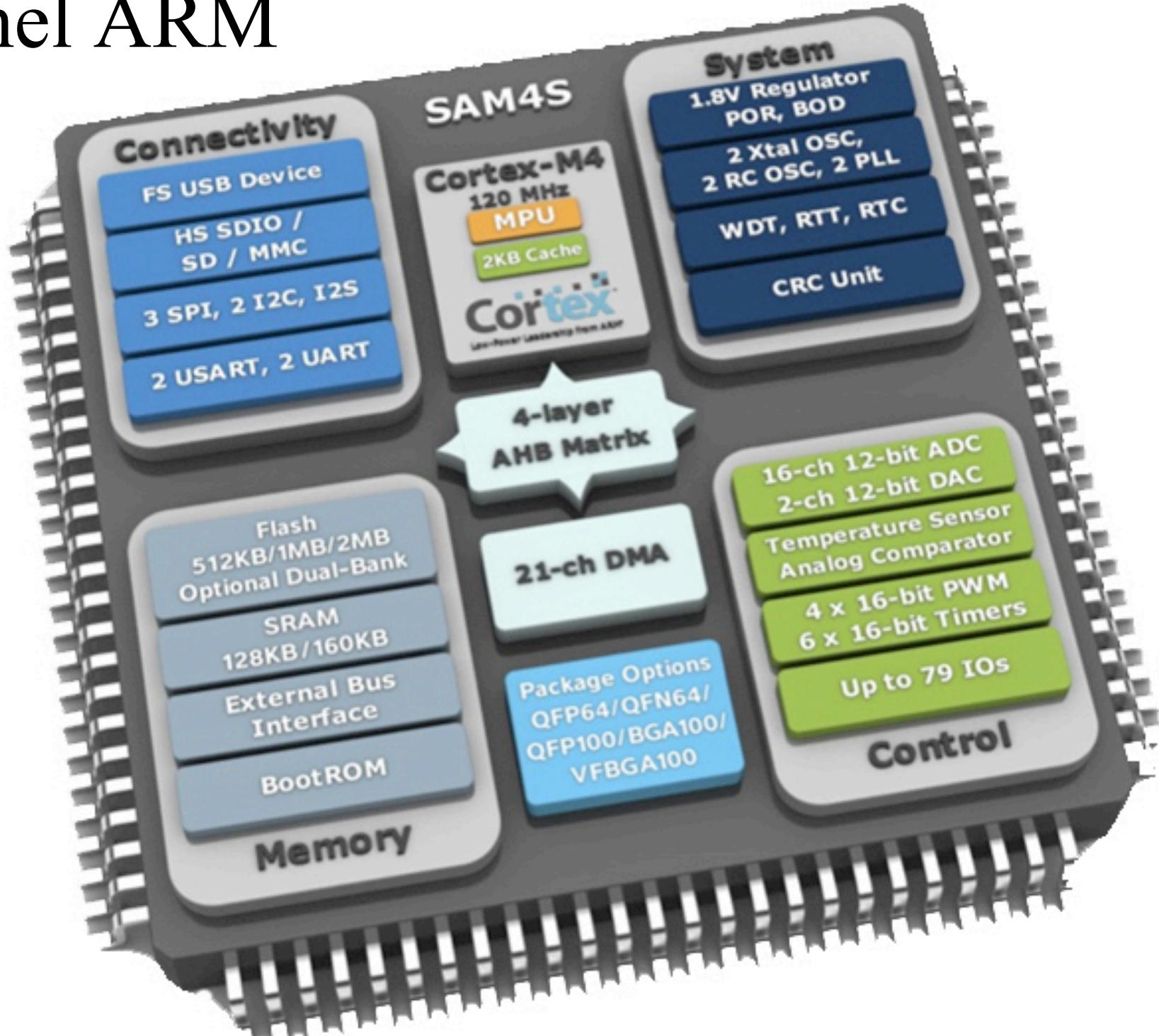


Arduino DUE with ARM Processor



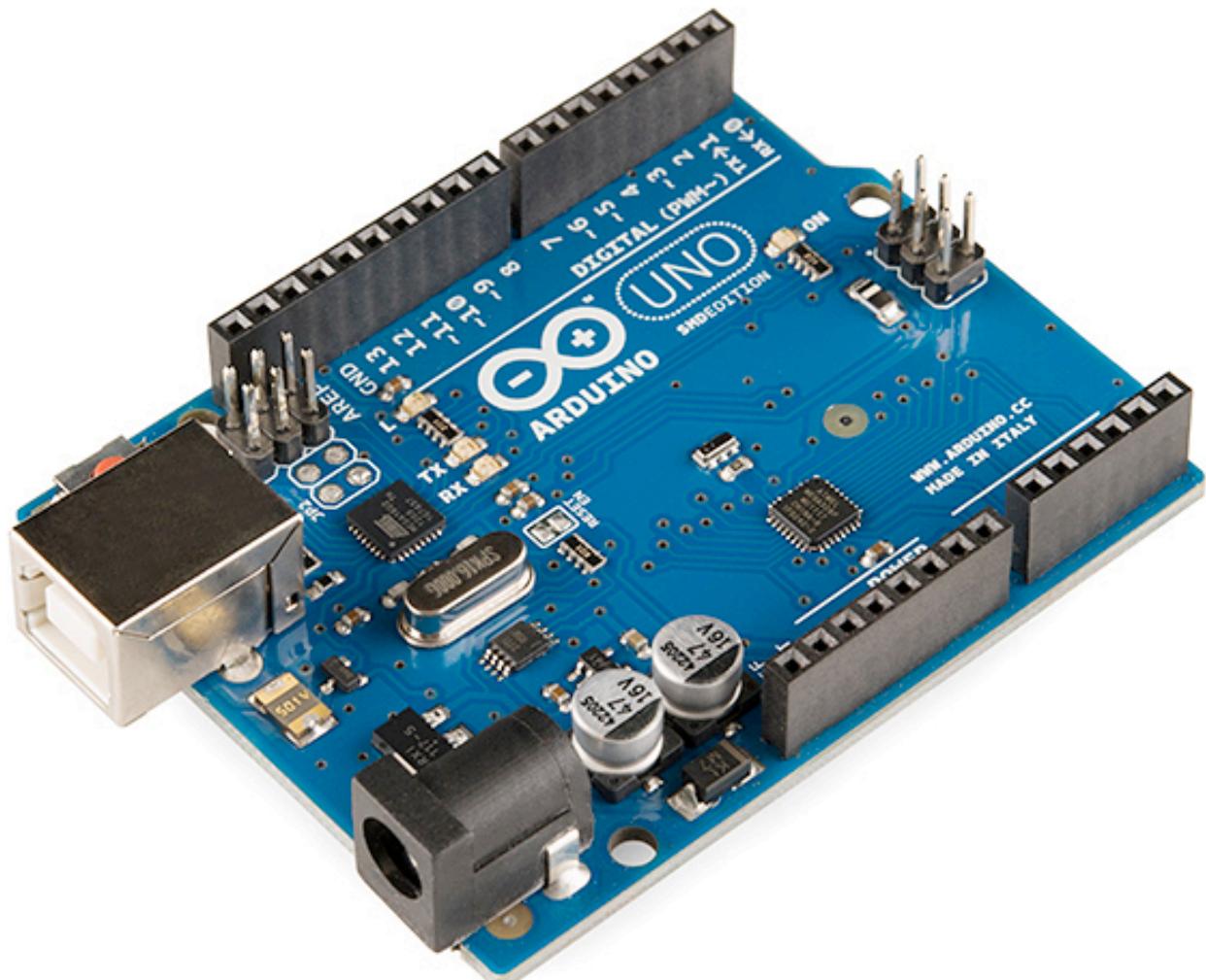
Based on the ARM® Cortex™-M3 processor, the Atmel® SAM3X8E runs at 84MHz and features 512KB of Flash and 100KB of SRAM.

Atmel ARM



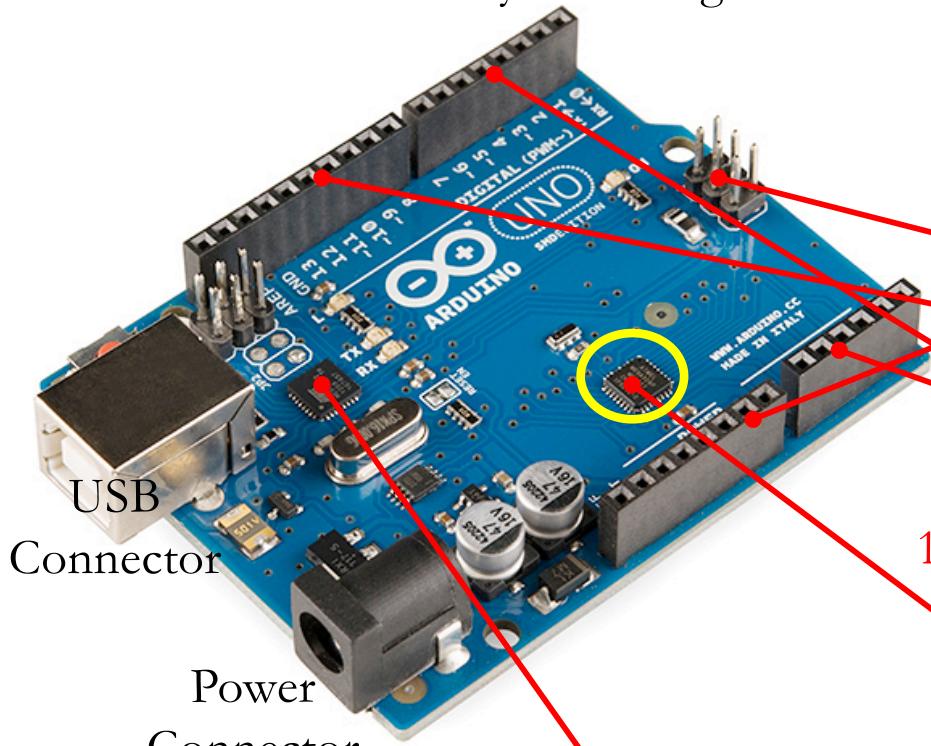
Arduino Uno

The **Arduino Uno** is a single-board microcontroller based on an open-source hardware board designed around an 8-bit Atmel AVR microcontroller.

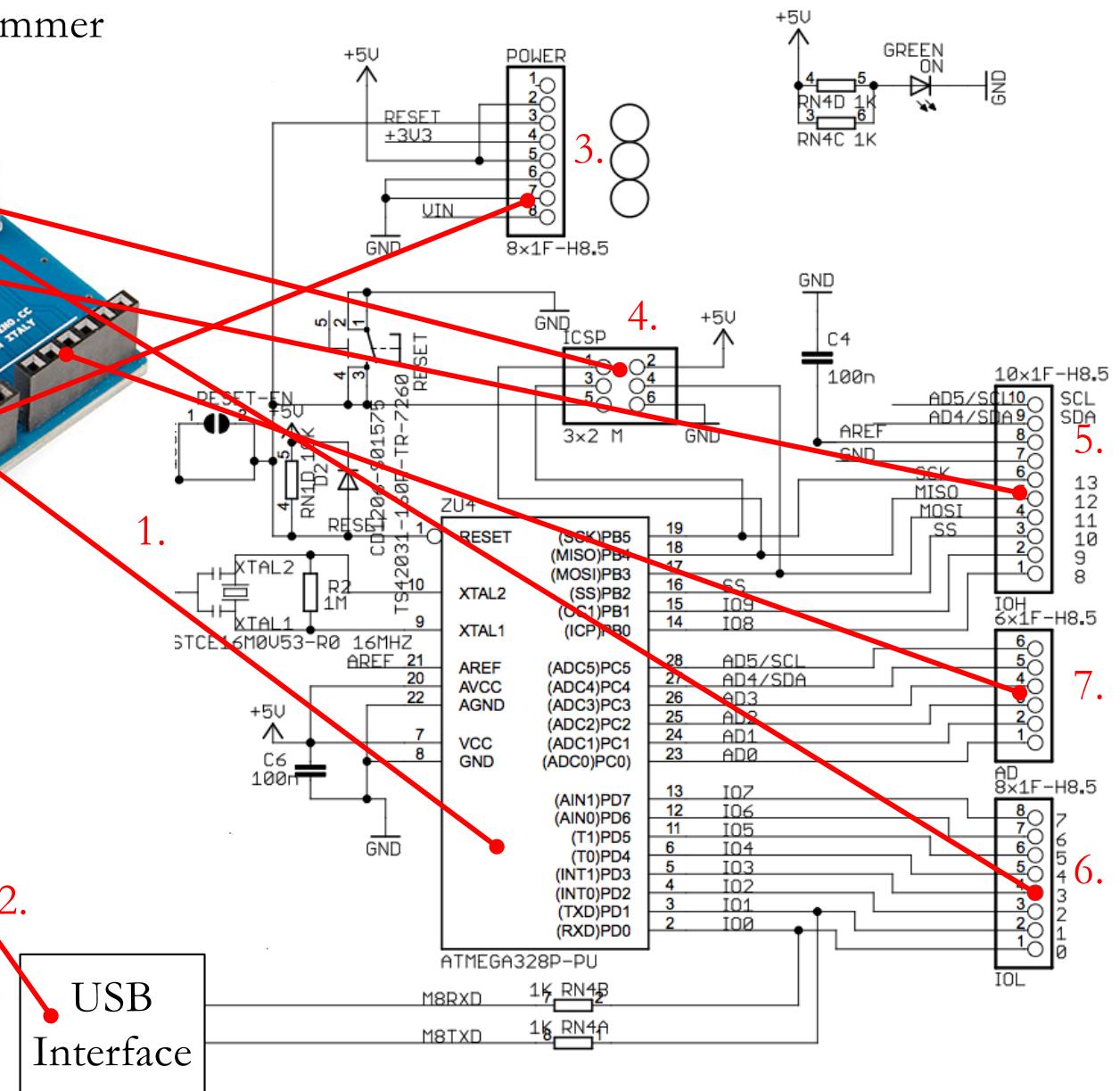


Simple, cheap, and easy to use.

1. AVR Microcontroller
2. USB Interface
3. Power Connector
4. ICSP In-Circuit System Programmer

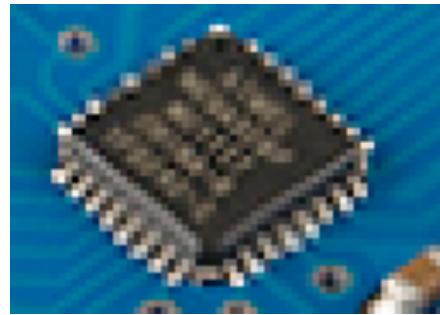


Arduino Schematic



Central Processing Unit (CPU)

The part of the microcontroller that executes the program.

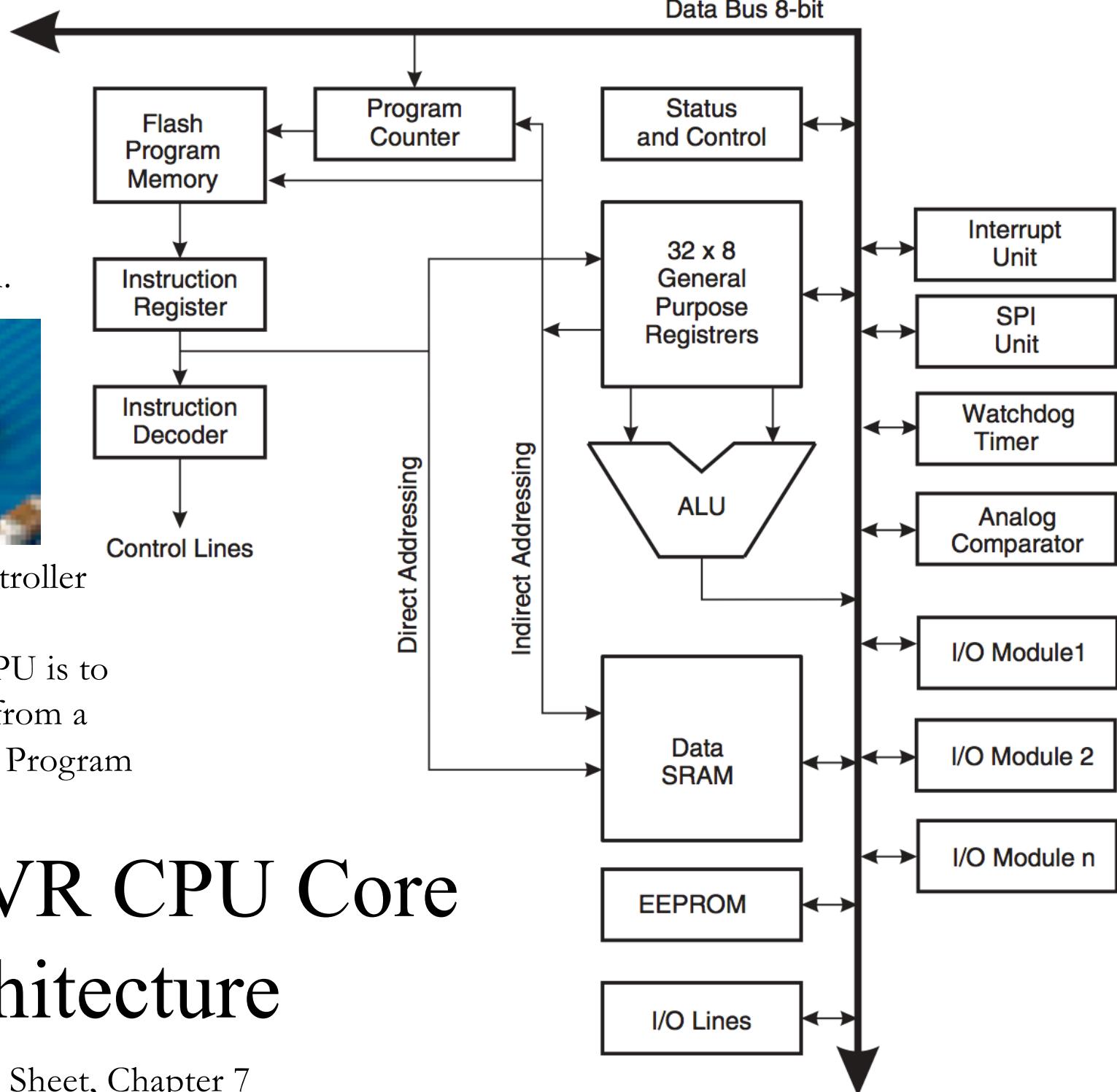


8-bit AVR MicroController

The purpose of the CPU is to execute instructions from a program stored in Flash Program Memory.

Atmel AVR CPU Core Architecture

Data Sheet, Chapter 7



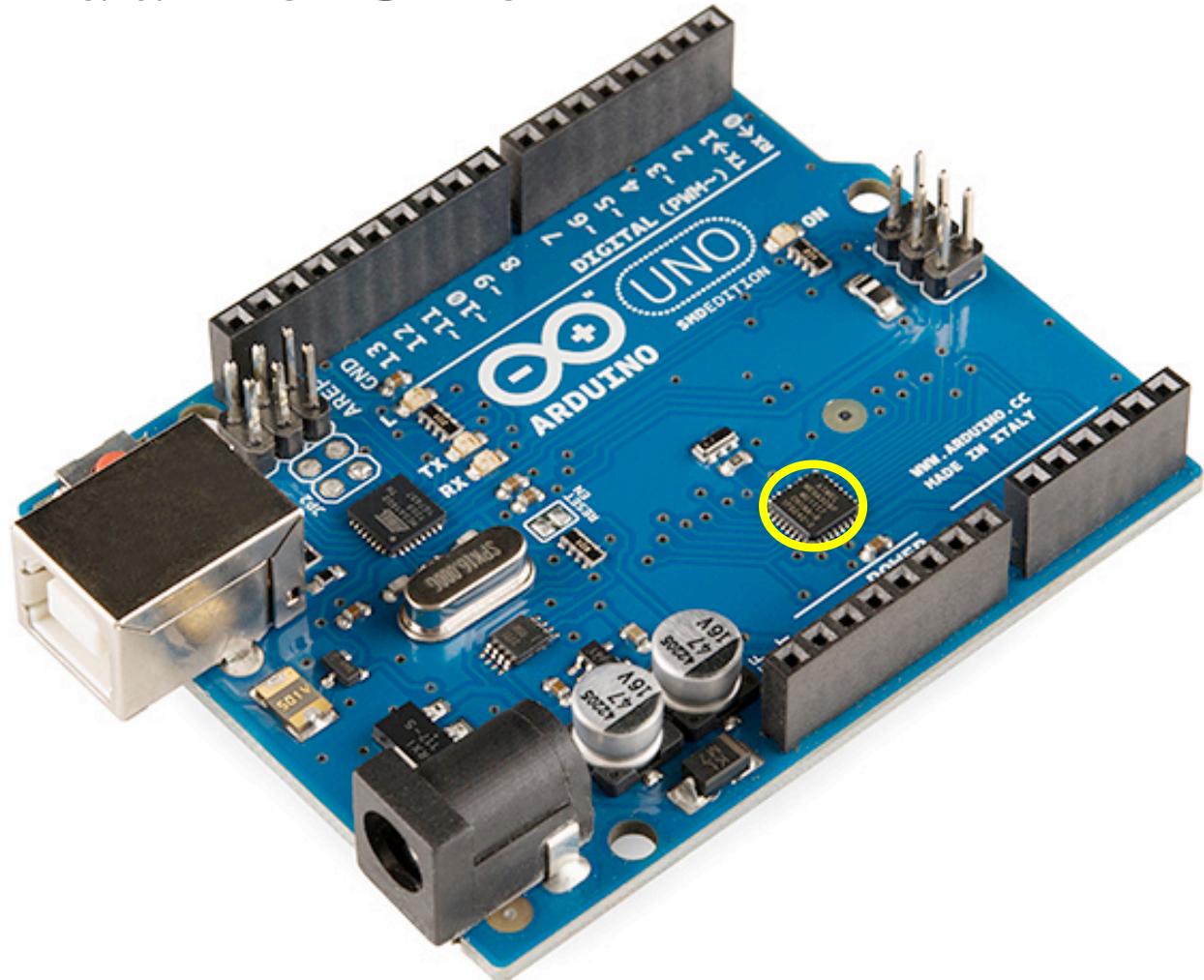
Lesson 23

Atmel AVR Processor Architecture
The C Language

Arduino Uno

The **Arduino Uno** is a single-board microcontroller based on an open-source hardware board designed around an 8-bit Atmel AVR microcontroller.

Simple, cheap, and easy to use.

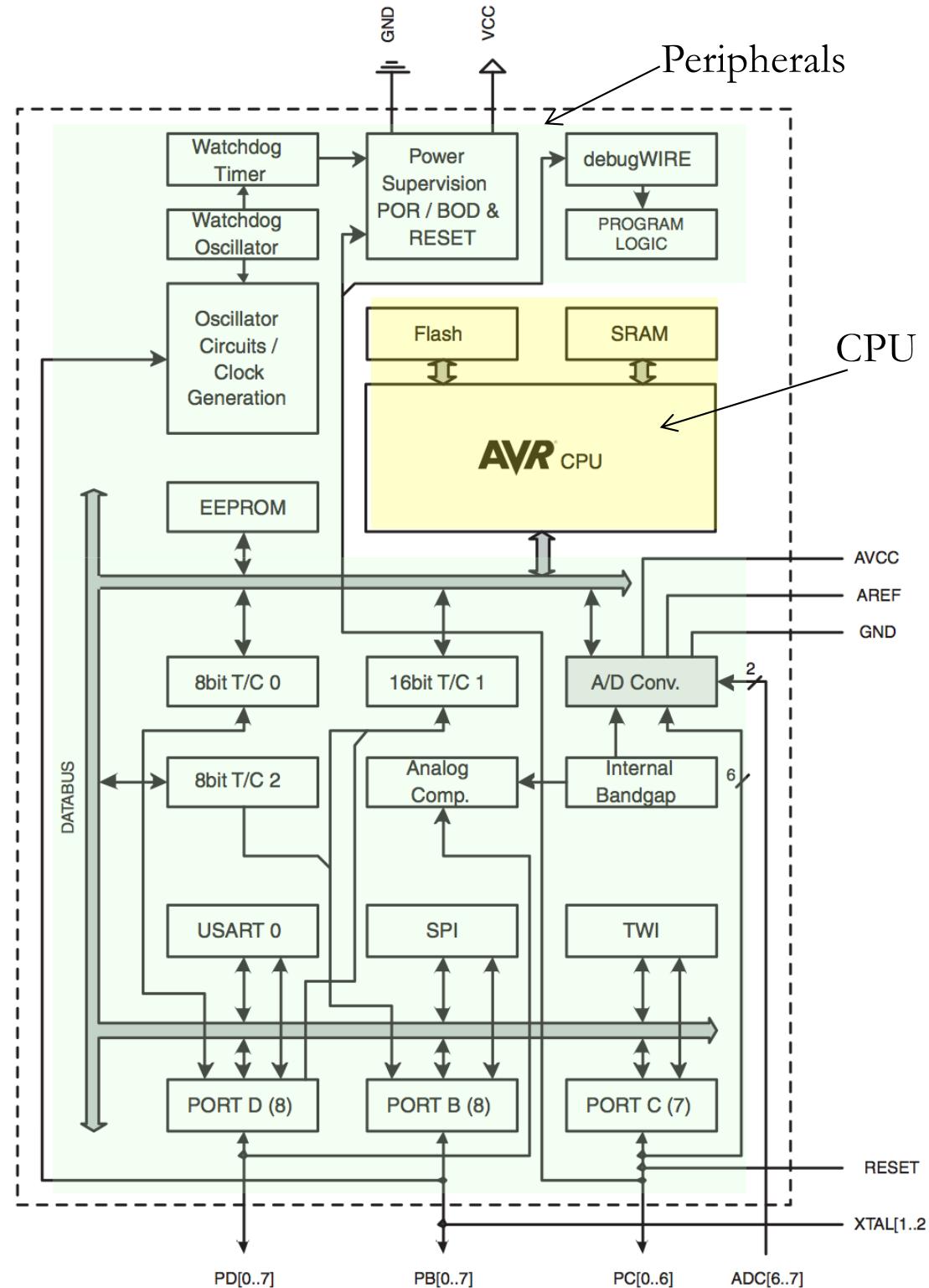


A low power 8-bit AVR RISC-based microcontroller with 32KB flash memory, 1KB EEPROM, and 2KB SRAM.

Block Diagram of the ATmega 328P

Peripherals:

1. EEPROM
2. Three T/C – Timers/Counters
3. Six PWM Channels
4. 10-bit A/D – Analog to Digital
5. USART – Serial Interface
6. SPI – Serial Interface
7. TWI (I²C) – Serial Interface
8. Watchdog Timer
9. Ports B, C, D – General Purpose I/O
10. Interrupt on pin change



Assembly Language Programming

- Architecture-specific low-level language executed on the CPU
- To program in Assembly, you have to understand the underlying CPU architecture to use it.
 - Registers and memory model
 - How memory is addressed
 - CPU Level Instructions
- Types of Instructions
 - Arithmetic/Logical (operands and results in the register file)
 - Load memory (value in memory is read and put in reg file)
 - Store memory (value from reg file is written to memory)
 - I/O (value from reg file is read from or written to peripheral)

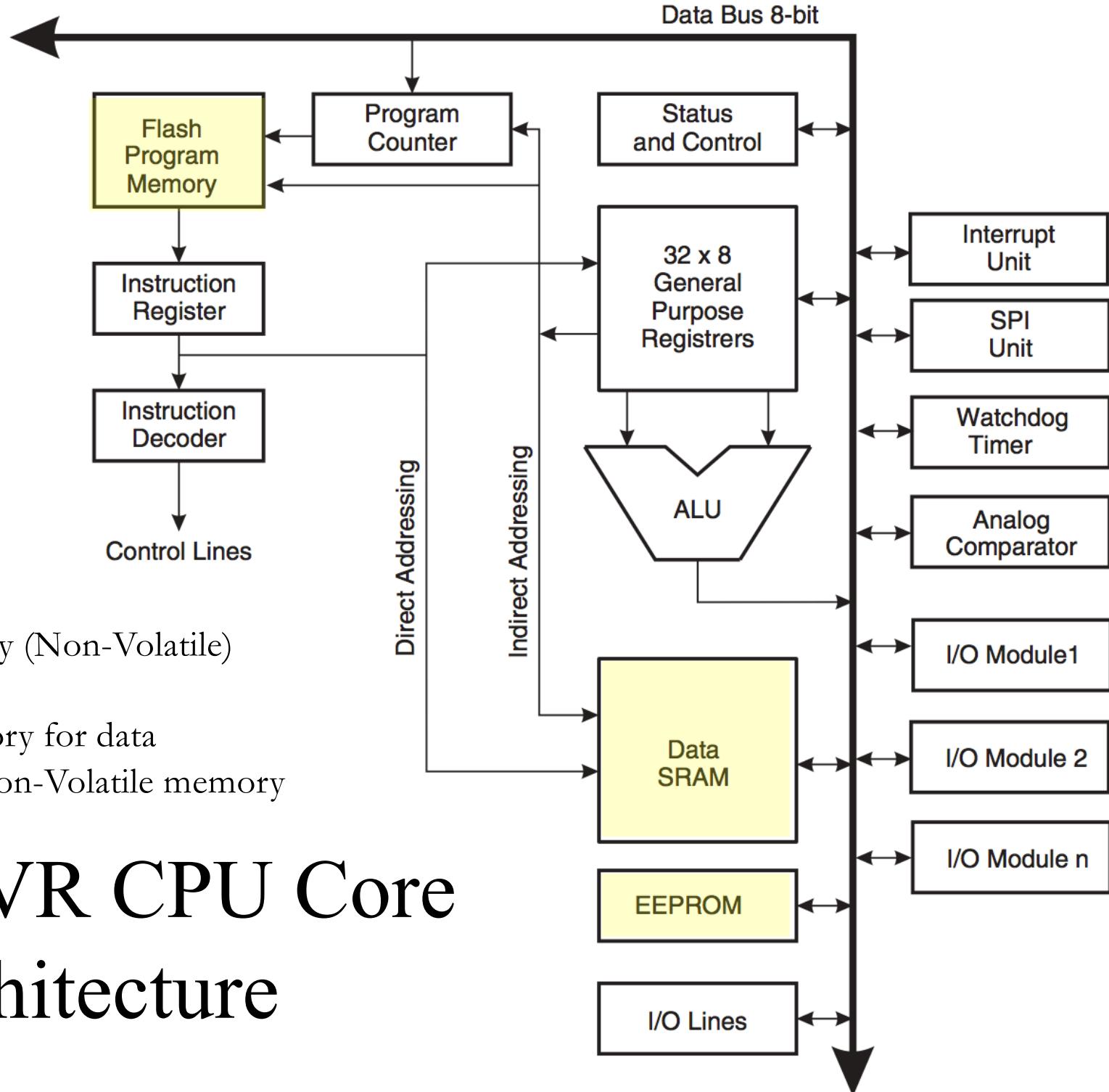
This is the Atmel AVR CPU that is used by the Arduino Uno.

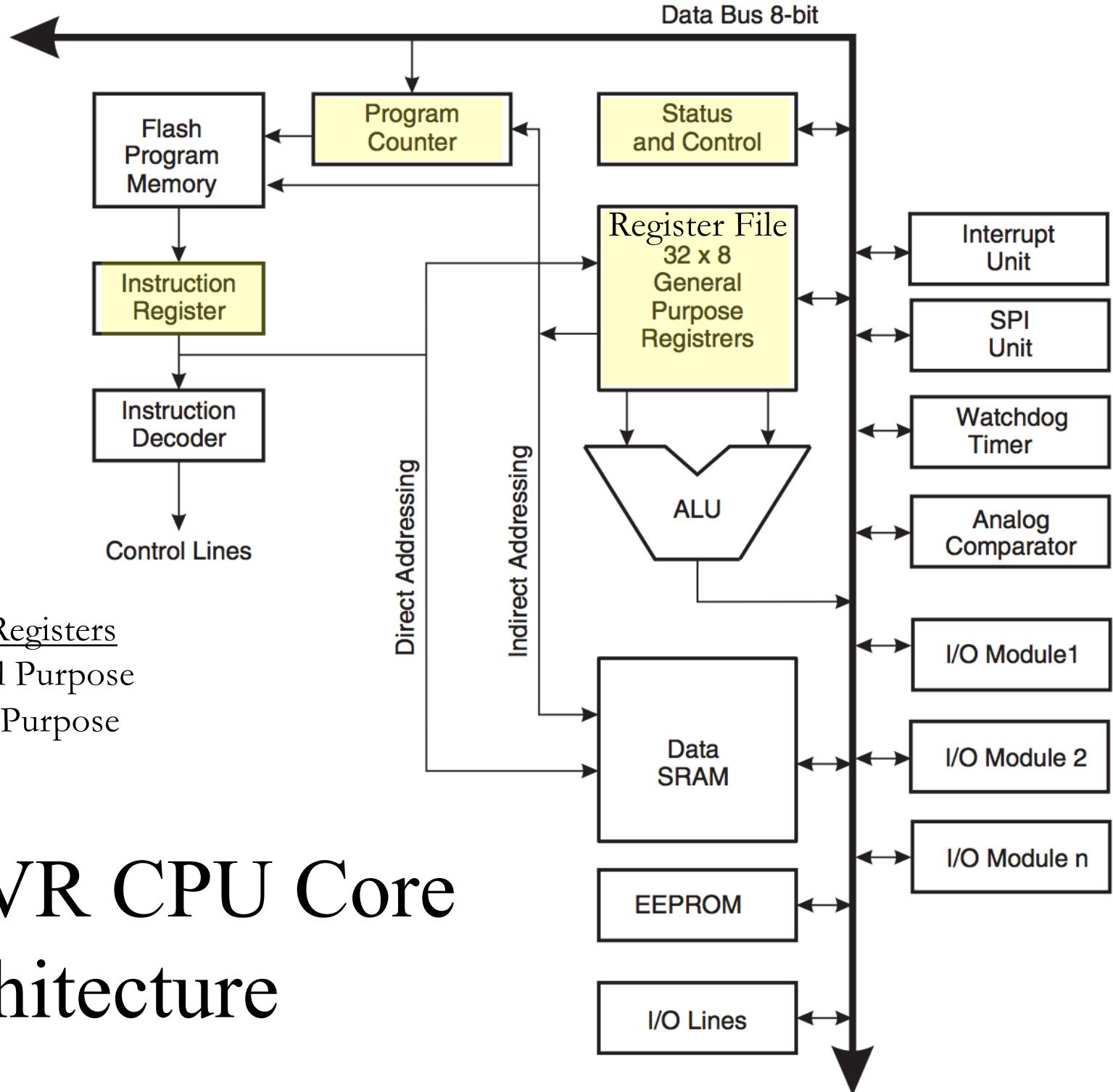
Let's take a look around the insides of the CPU and see what kind of things we can recognize.

Memory

1. Flash Memory (Non-Volatile) for programs
2. SRAM Memory for data
3. EEPROM Non-Volatile memory

Atmel AVR CPU Core Architecture

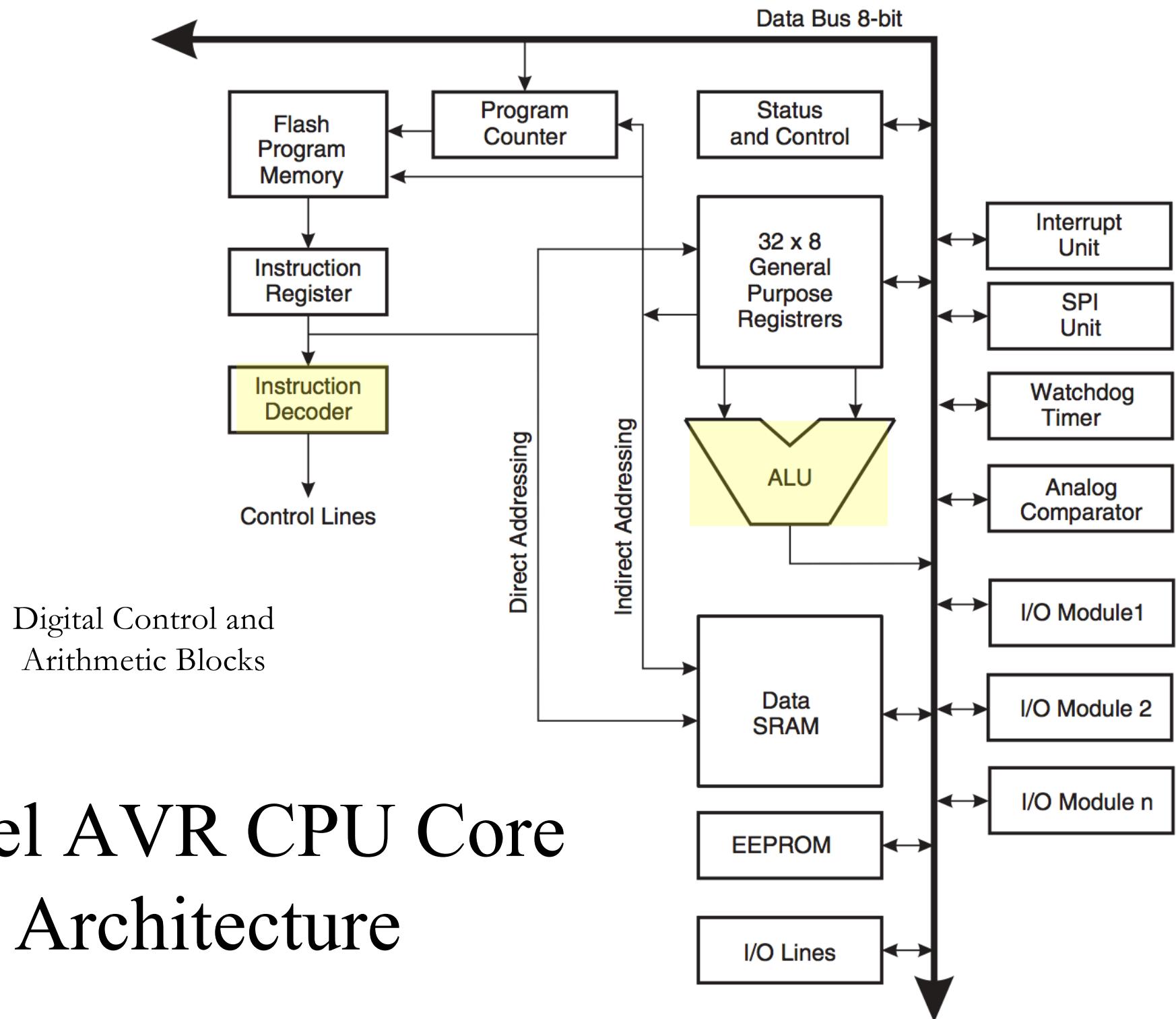




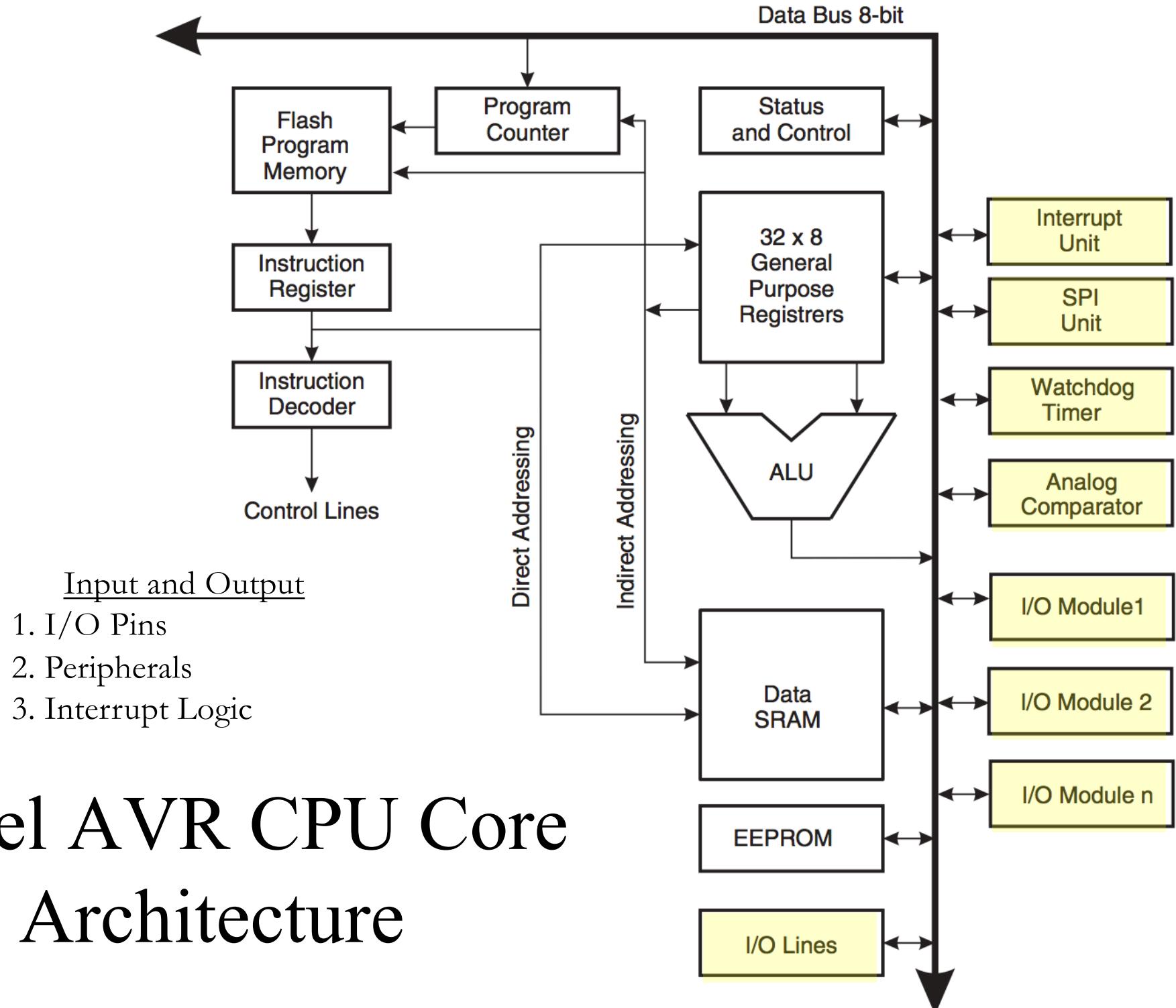
Registers

1. General Purpose
2. Special Purpose

Atmel AVR CPU Core Architecture



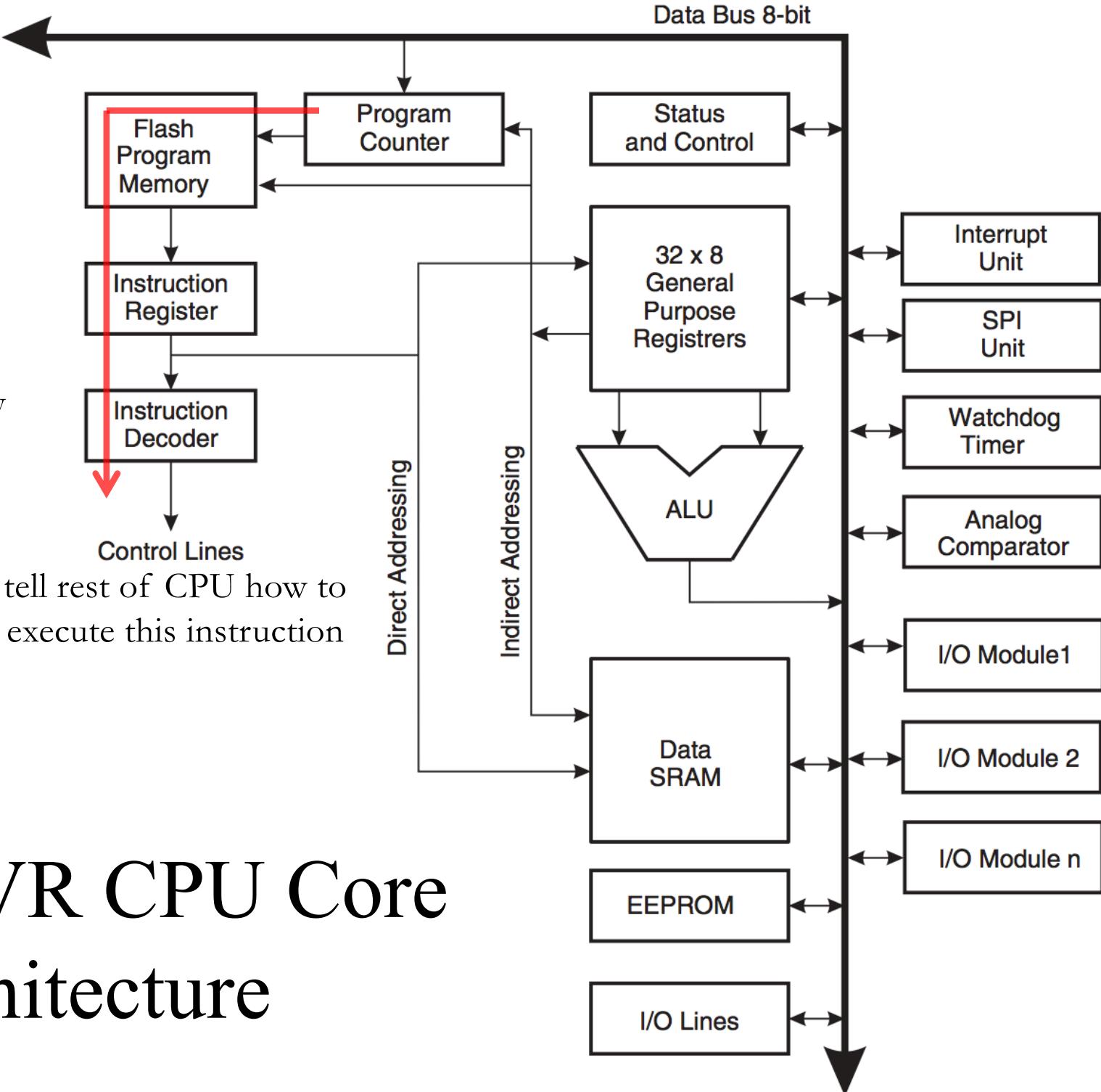
Atmel AVR CPU Core Architecture



Atmel AVR CPU Core Architecture

How low-level instructions are executed in the CPU

1. Load instruction from Flash memory
2. Decode instruction
3. Execute instruction

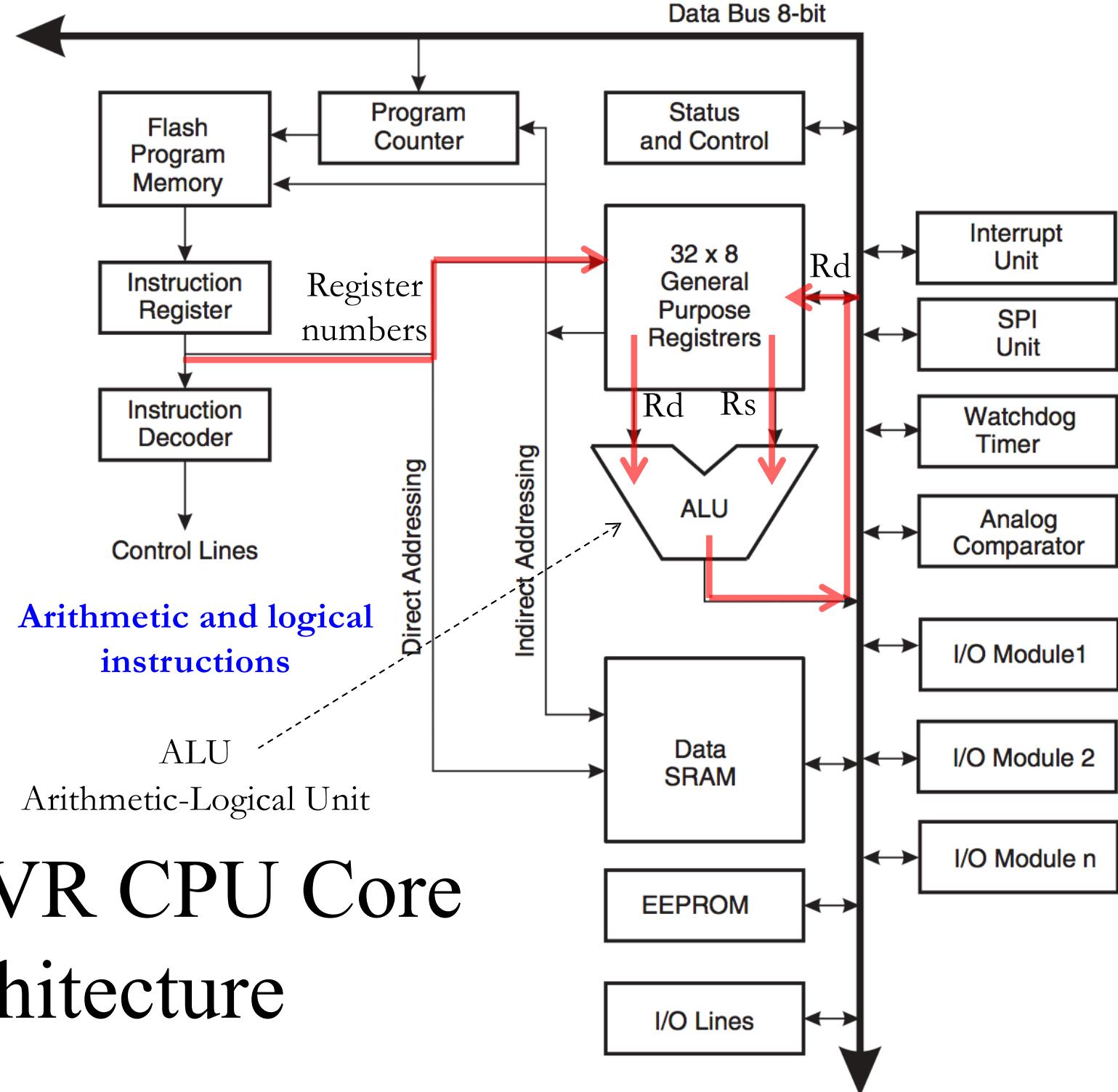


Atmel AVR CPU Core Architecture

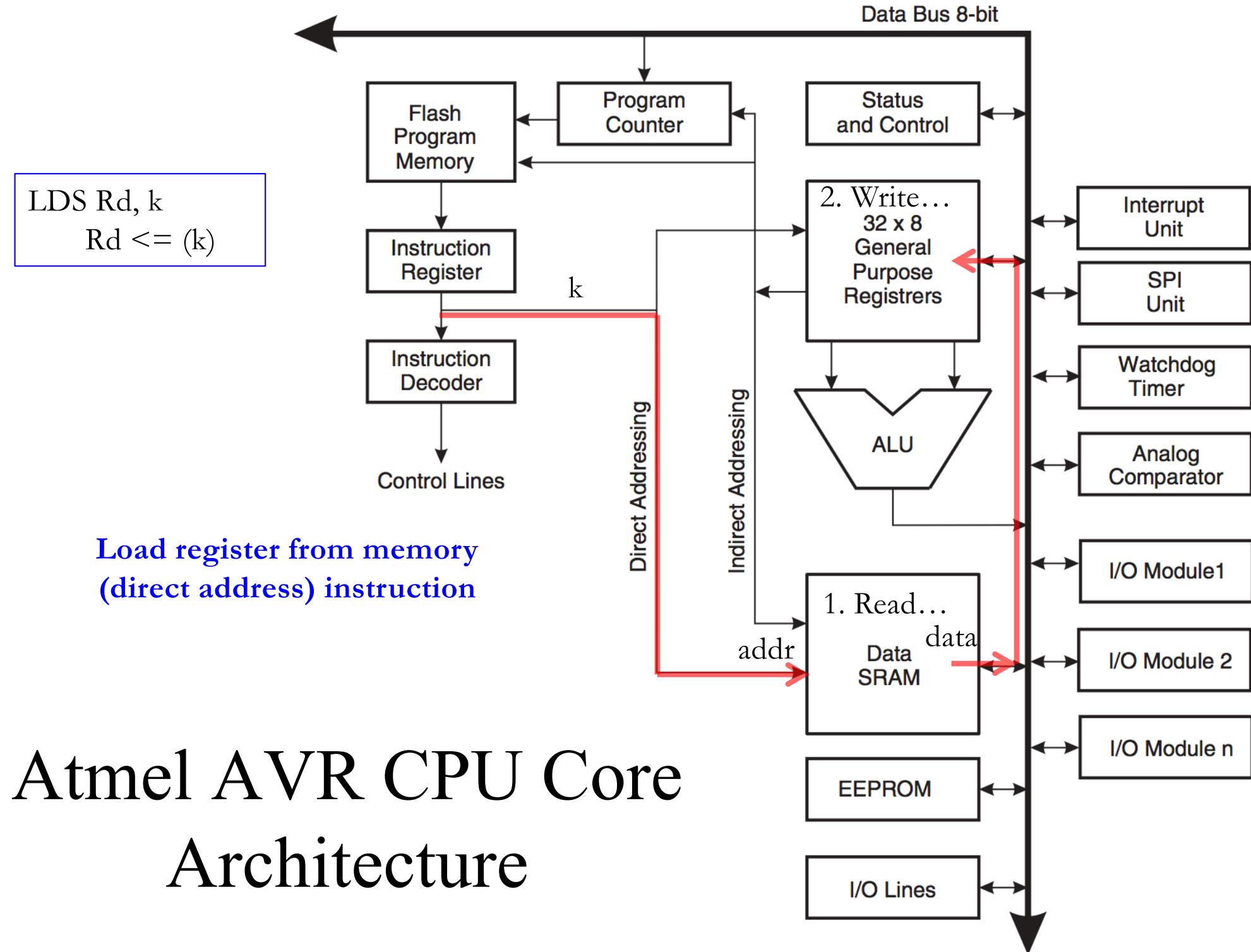
```

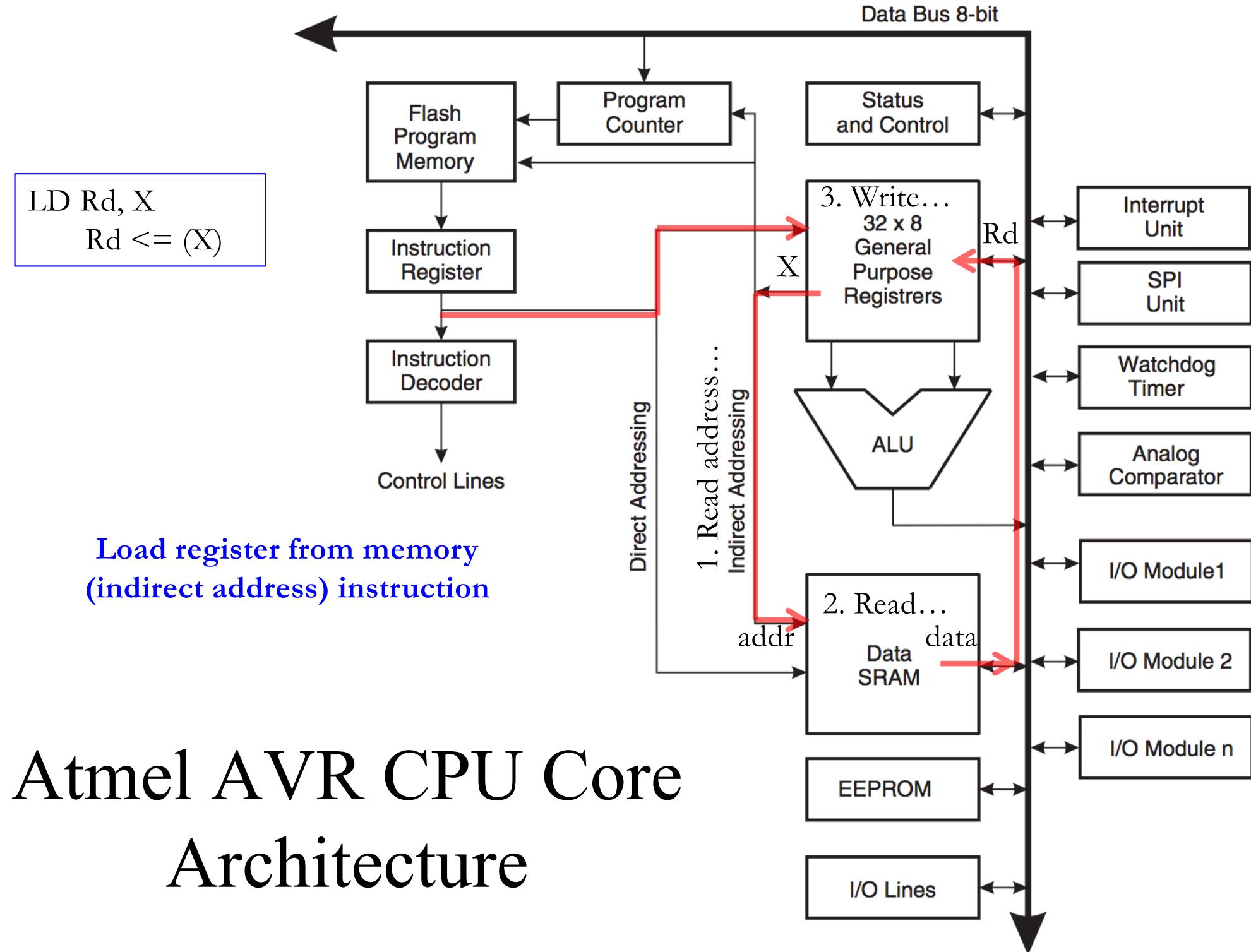
ADD Rd, Rs
    Rd <= Rd + Rs
SUB Rd, Rs
    Rd <= Rd - Rs
MUL Rd, Rs
    Rd <= Rd * Rs
AND Rd, Rs
    Rd <= Rd & Rs
OR  Rd, Rs
    Rd <= Rd | Rs
EOR Rd, Rs
    Rd <= Rd ^ Rs
COM Rd
    Rd <= ~Rd
NEG Rd
    Rd <= -Rd
INC Rd
    Rd <= Rd + 1
DEC Rd
    Rd <= Rd - 1

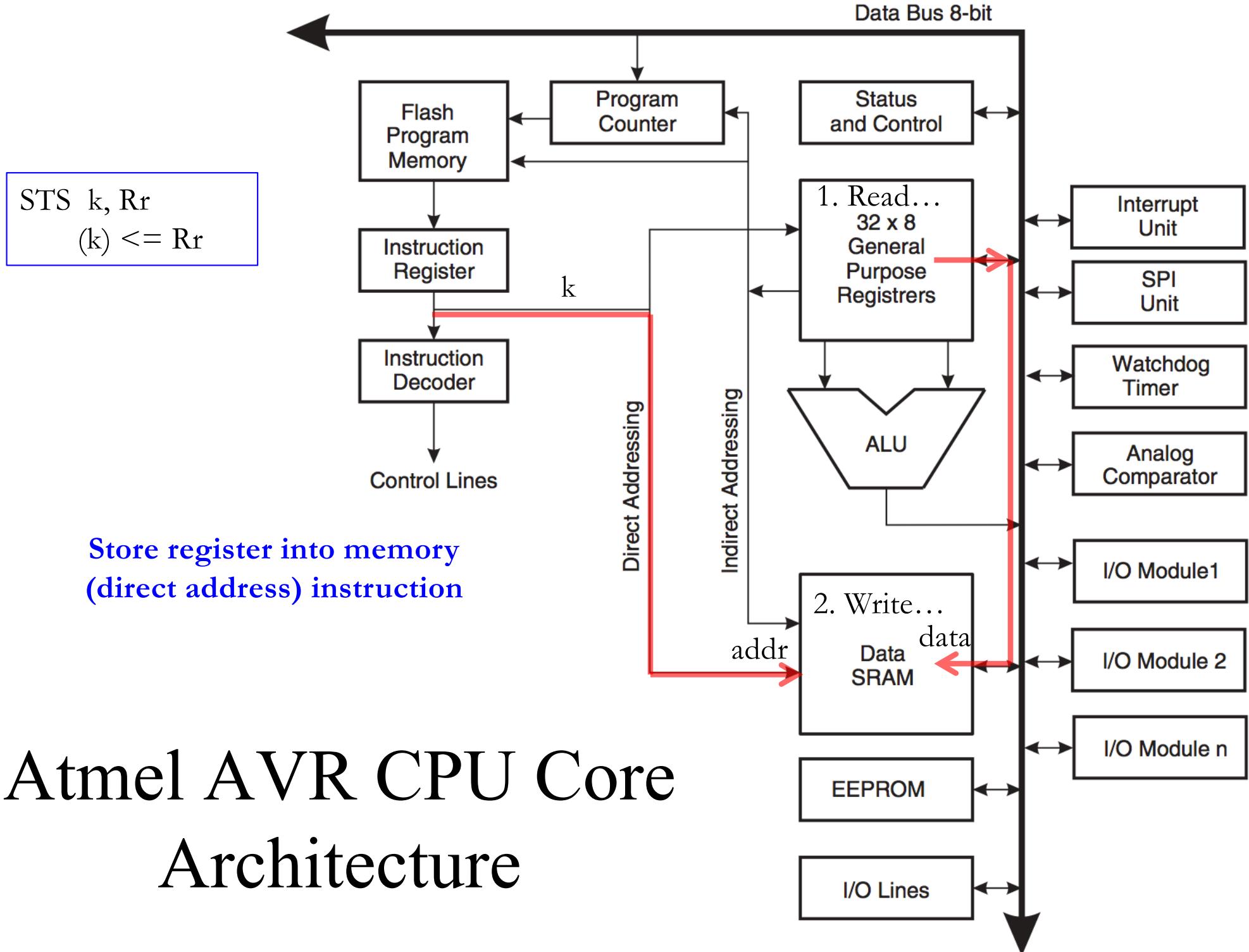
```

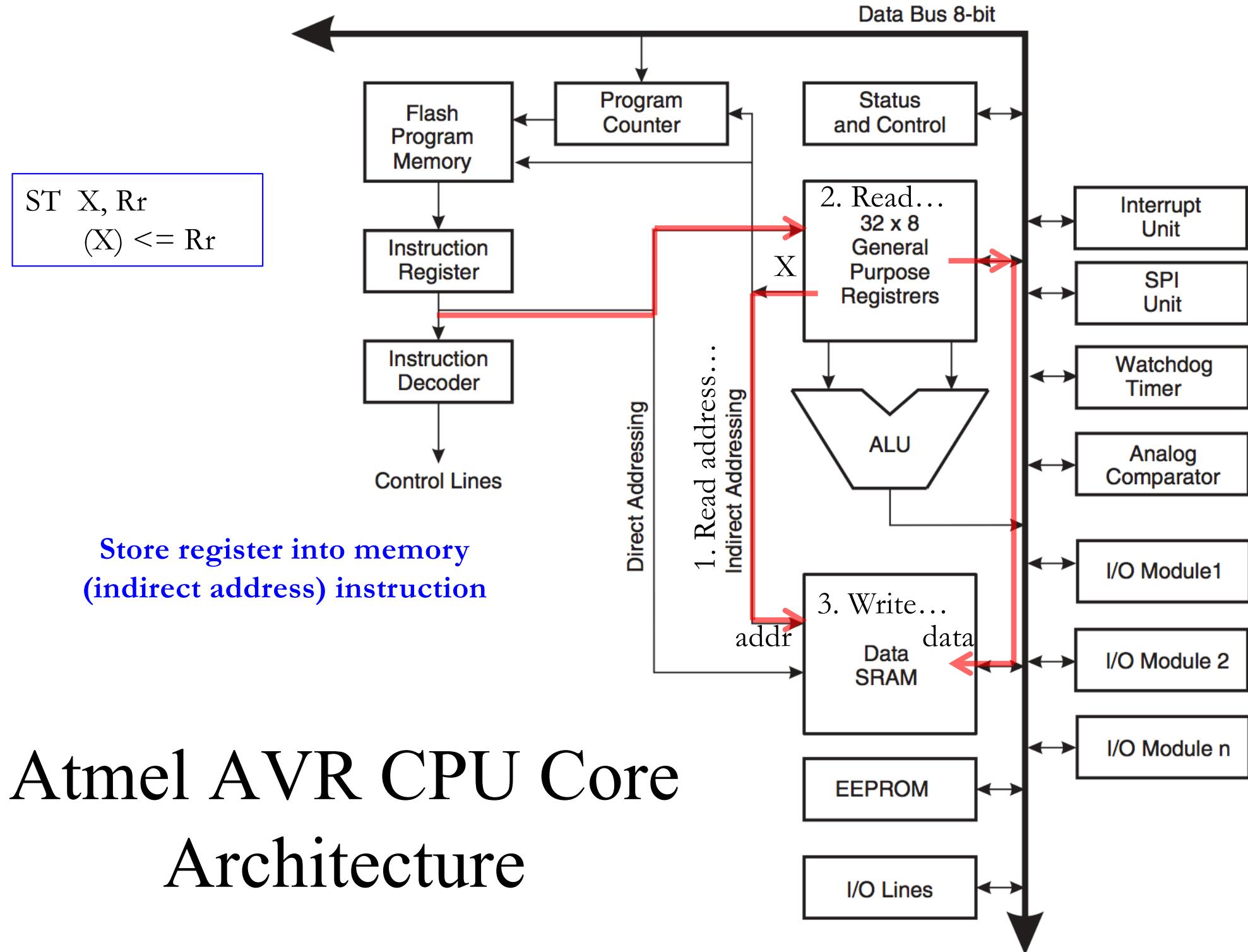


Atmel AVR CPU Core Architecture







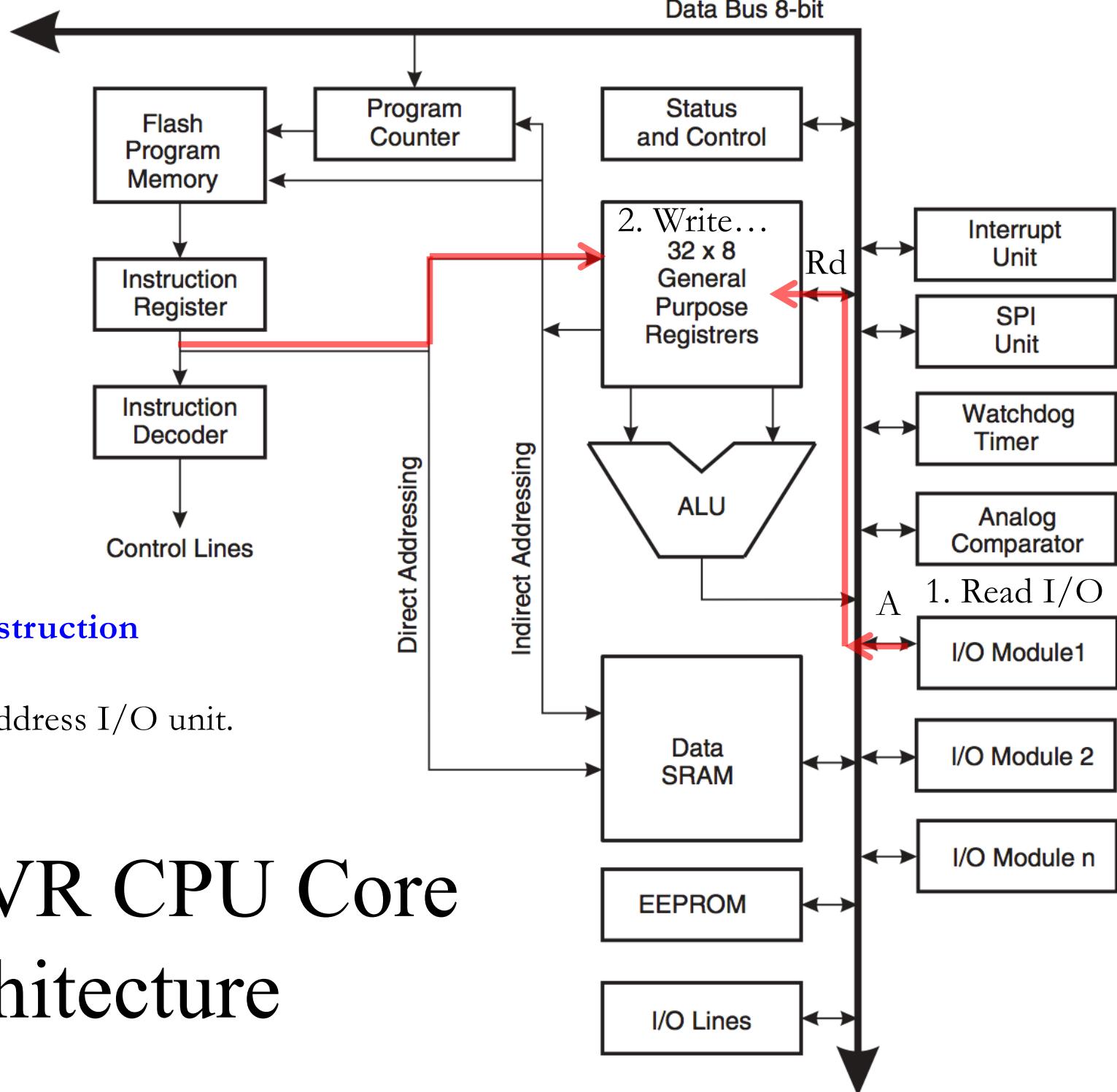


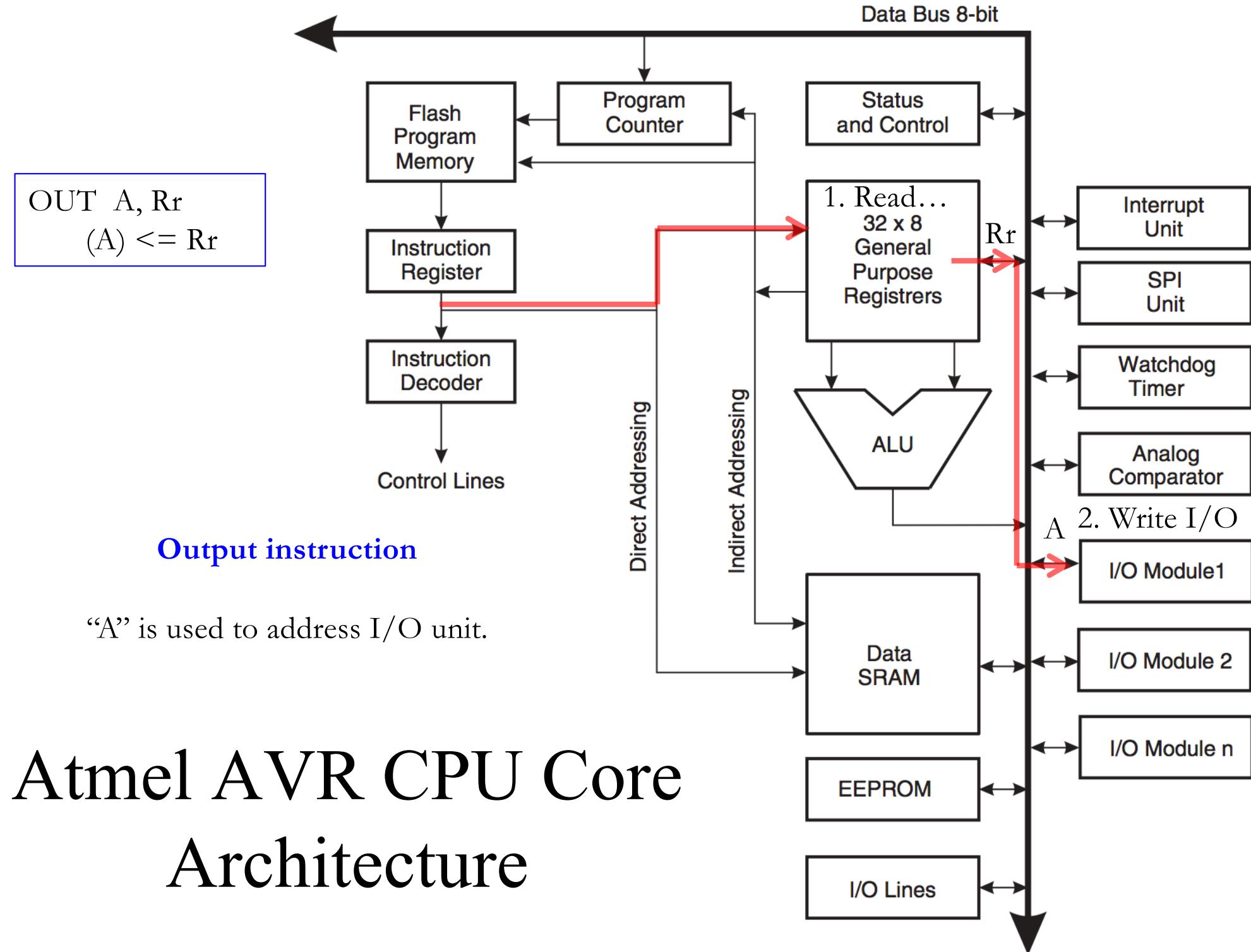
IN Rd, A
Rd <= (A)

Input instruction

“A” is used to address I/O unit.

Atmel AVR CPU Core Architecture



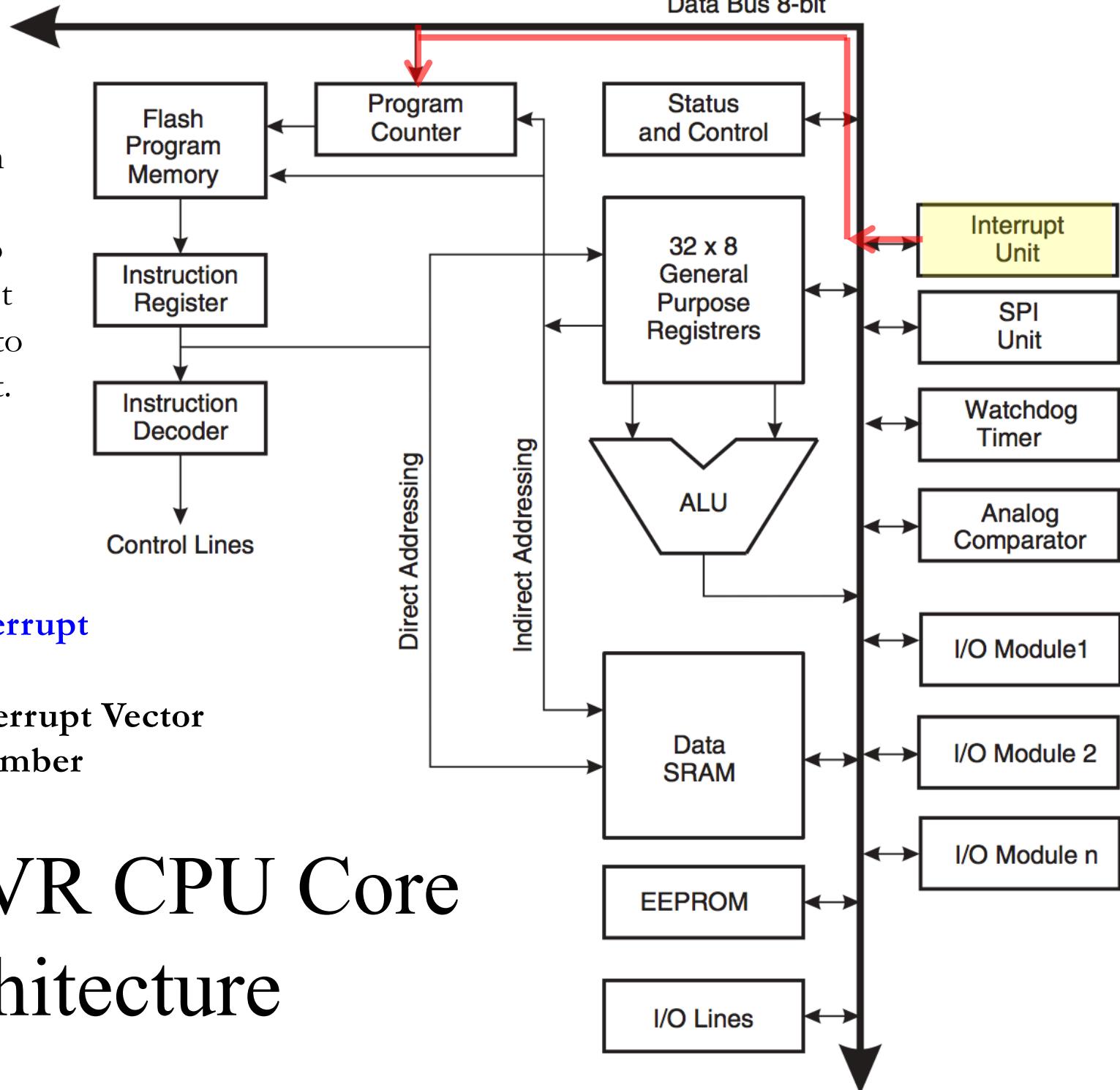


An “interrupt” is an external event that requires the CPU to execute an “interrupt handler subroutine” to respond to the event.

Interrupt

$\text{PC} \leq \text{Interrupt Vector Number}$

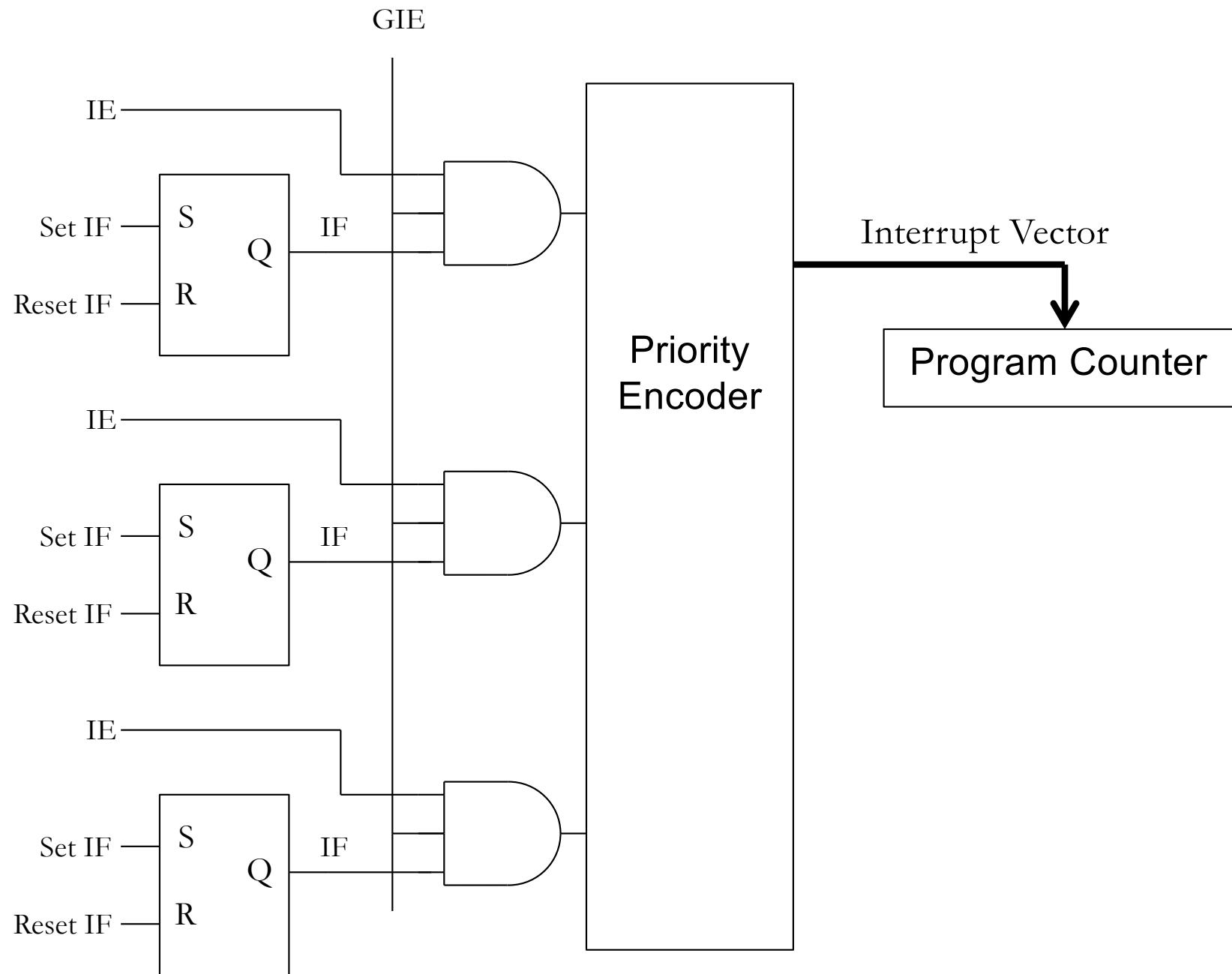
Atmel AVR CPU Core Architecture



Interrupts

- Interrupt flags are set when a certain event happens that requires attention from the program.
- Global Interrupt Enable (GIE) must be 1 for any interrupts to happen. The specific Interrupt Enable (IE) for each kind of interrupt must also be 1 for that specific interrupt to happen.
- Each interrupt has an entry in the interrupt vector table. In the AVR, when an interrupt occurs that is enabled (both GIE and IE are 1), then the program executes the instruction in the specified interrupt vector table entry. (That instruction is usually a jump to the interrupt handler subroutine.)
- The interrupt handler resets the interrupt flag, services the interrupt, and then returns back to the regular code.

Interrupt Controller



Assembly Language Programming

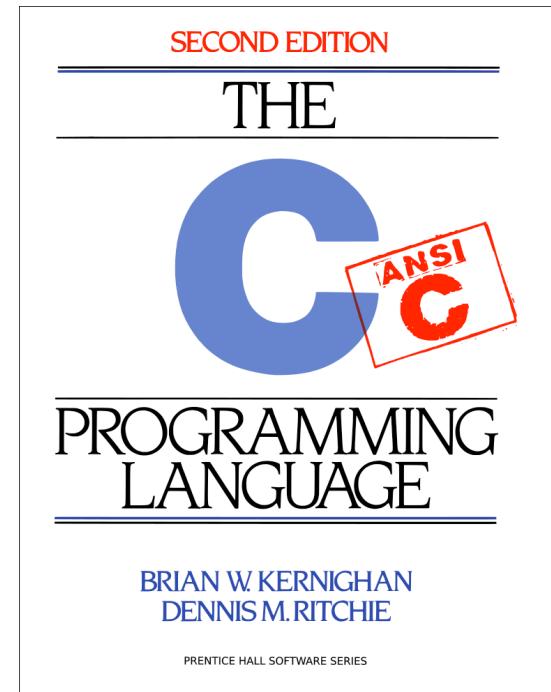
- Architecture-specific low-level language executed on the CPU
- Programmer must manage memory
- Difficult and error prone
- Not portable between different computer architectures.

High Level Languages

- Allow us to use symbolic names for variables
- Programmer simply assigns each value a name and doesn't worry about the details of how memory is managed.
- The compiler takes care of ...
 - register usage
 - choosing where variables are stored
 - loads and stores from memory
 - how code gets executed

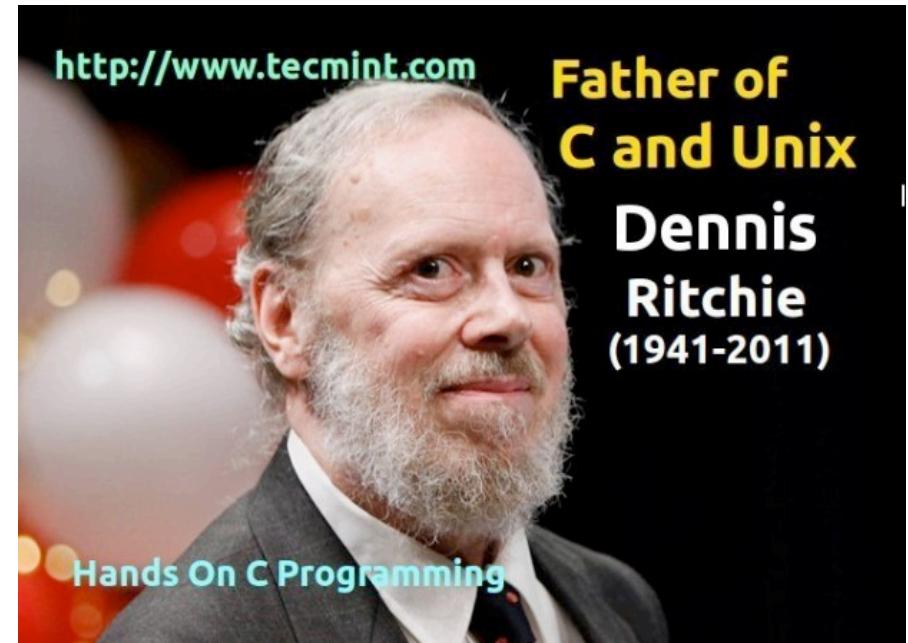
The C Language

- The most commonly-used language for programming embedded systems
- “High-level assembly”
 - Almost as powerful as assembly
 - but is a simple high-level language
- Very portable: compilers exist for virtually every processor and microcontroller
- Easy to learn and use
- Produces efficient code



C Language History

- Developed at Bell Labs along with Unix OS
- Developed from 1969 to 1973. 1972 first release.
- Due mostly to Dennis Ritchie
- Designed for systems programming
 - Operating systems
 - Utility programs
 - Compilers
- C is the predecessor to most of modern languages such as C++ and Java.



C Language

- Provides an abstraction of underlying hardware
 - Hides low level architecture details from programmer
 - General programming language not tied to architecture
 - Portable programs, work on almost all architectures
 - The compiler generates the machine-level instructions.
- Provides expressiveness
 - Express complex tasks with small amount of code
 - Human-friendly, English-like and human readable

```
if (isCloudy)
    get(umbrella);
else
    get(sunglasses);
```

Summary

- Memory (RAM)
 - SRAM, DRAM – volatile types of memory
 - Flash, EEPROM – non-volatile types of memory
- Microcontrollers
 - CPU + peripherals on a single chip
- Arduino platforms, Uno and Due
- AVR Architecture (by Atmel Corp.)
 - How CPU executes instructions
- Programming Languages
 - Low-level machine assembly language
 - High-level C language

Lecture 24

Serial Interfaces

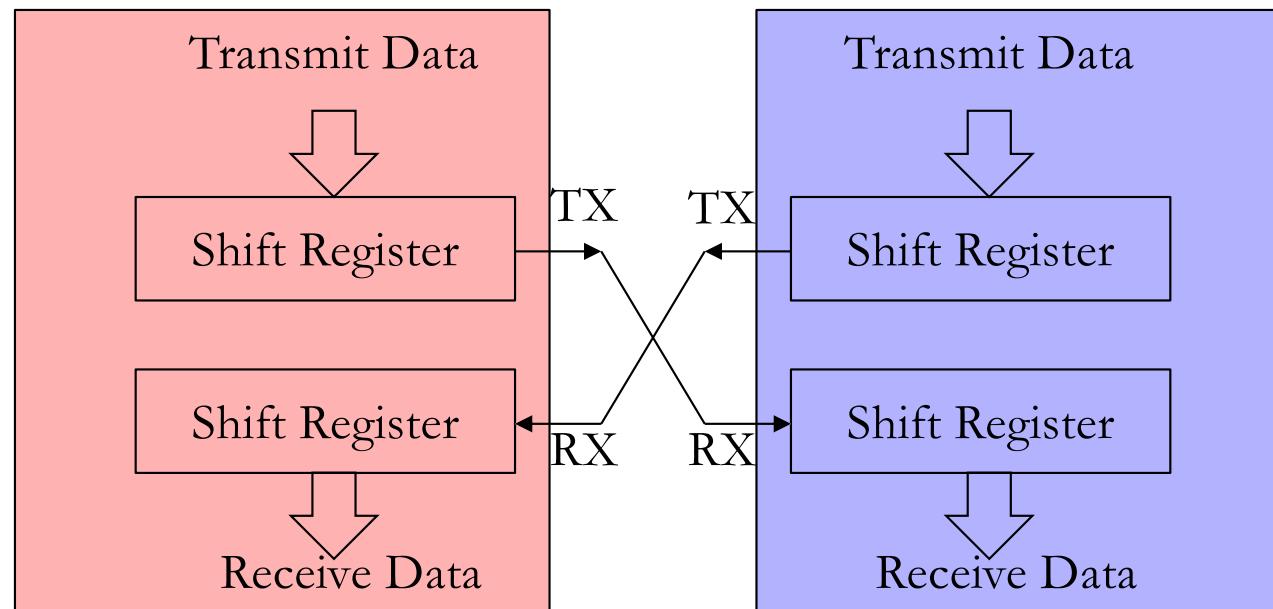
I2C, SPI, Interfacing to Sensors

Serial Communication Protocols

- Serial communication is the preferred way to transmit data because it doesn't use a lot of wires.
- **Synchronous** – Clock is sent with data.
Data is shifted and sampled using that clock.
 - SPI
 - I²C
- **Asynchronous** – No clock is sent. Sender and receiver must create their own clocks locally.
 - UART
 - USB

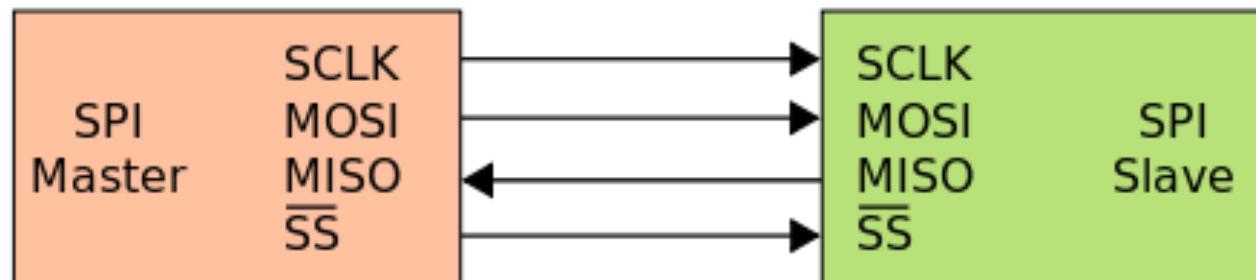
Serial Communication

- All serial communication is based on the shift register.
- The transmitter is a parallel-in serial-out shift register.
- The receiver is a serial-in parallel-out shift register.
- **Full Duplex** – data can be sent and received at the same time.
- **Half Duplex** – data can be sent in one direction at a time, but not both directions at the same time.



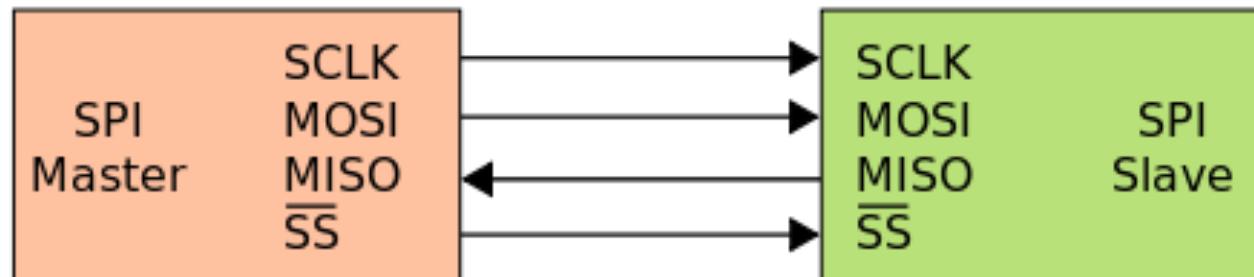
SPI (Serial Peripheral Interface)

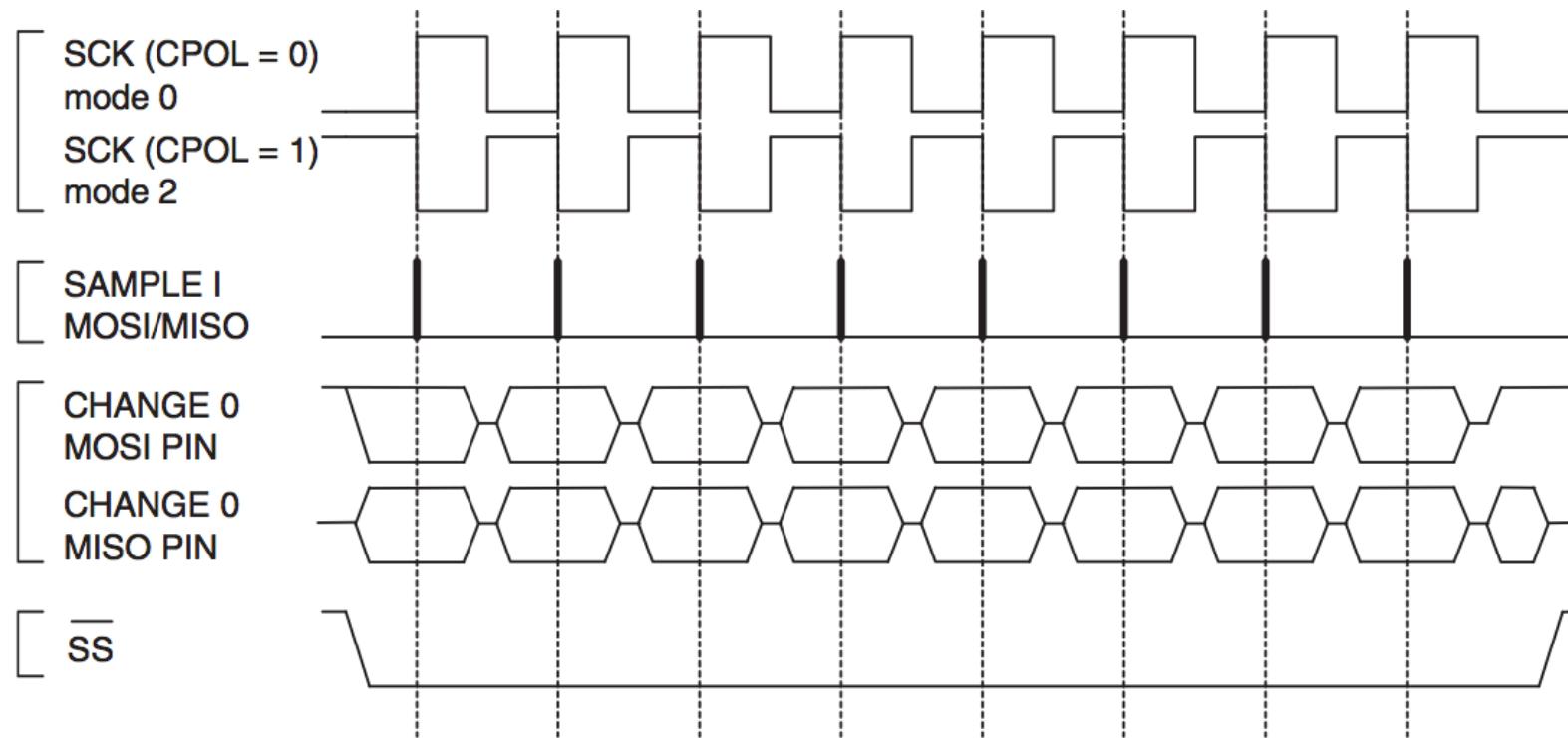
- Devices communicate in master/slave mode where the master device initiates and controls the timing of the transmission.
- Multiple slave devices are allowed (only one shown below).
- Synchronous (data shifts on clock edges, usually MSB first)
- Full duplex (data is transmitted in both directions at the same time).
- Very common microcontroller/sensor interface because of its simplicity and speed.



SPI Communication Signals

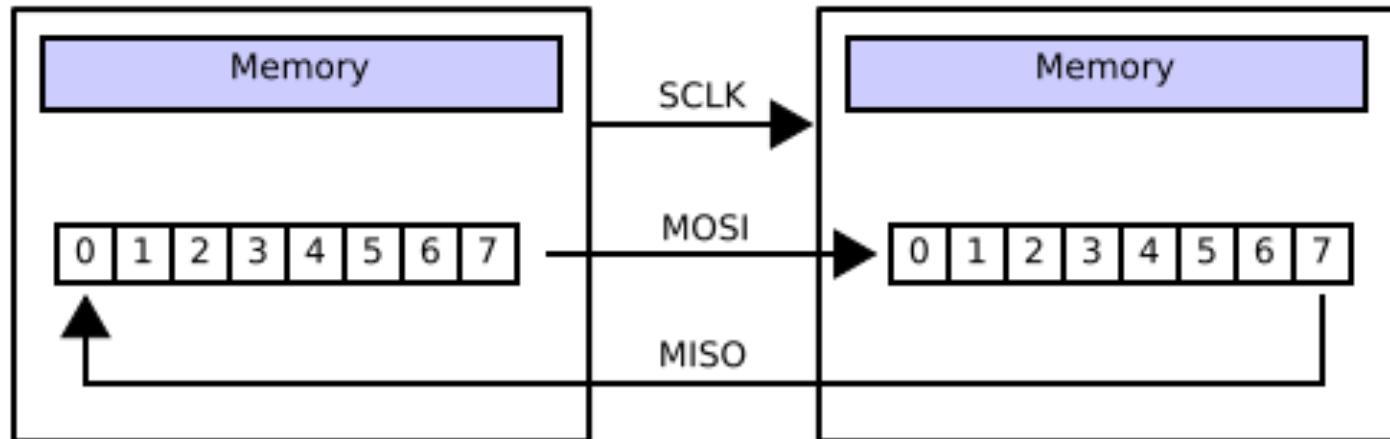
- The SPI bus specifies four logic signals:
 - SCLK : Serial Clock (output from master).
Clock edges used for shifting and sampling.
 - MOSI : Master Output, Slave Input
(Serial data output from master).
 - MISO : Master Input, Slave Output
(Serial data output from slave).
 - $\overline{\text{SS}}$: Slave Select (active low, output from master).
Falling edge starts a communication.





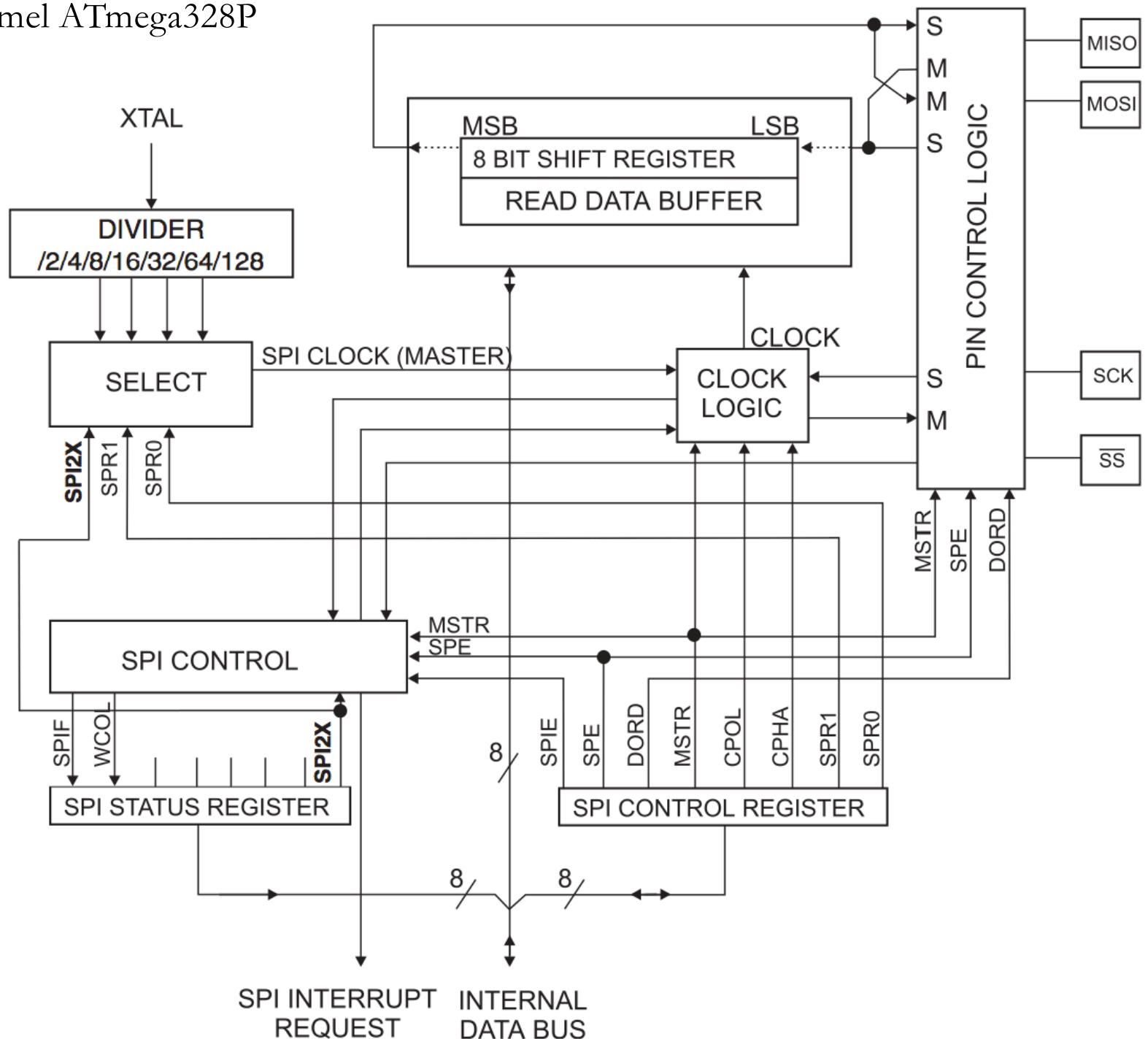
Master

Slave

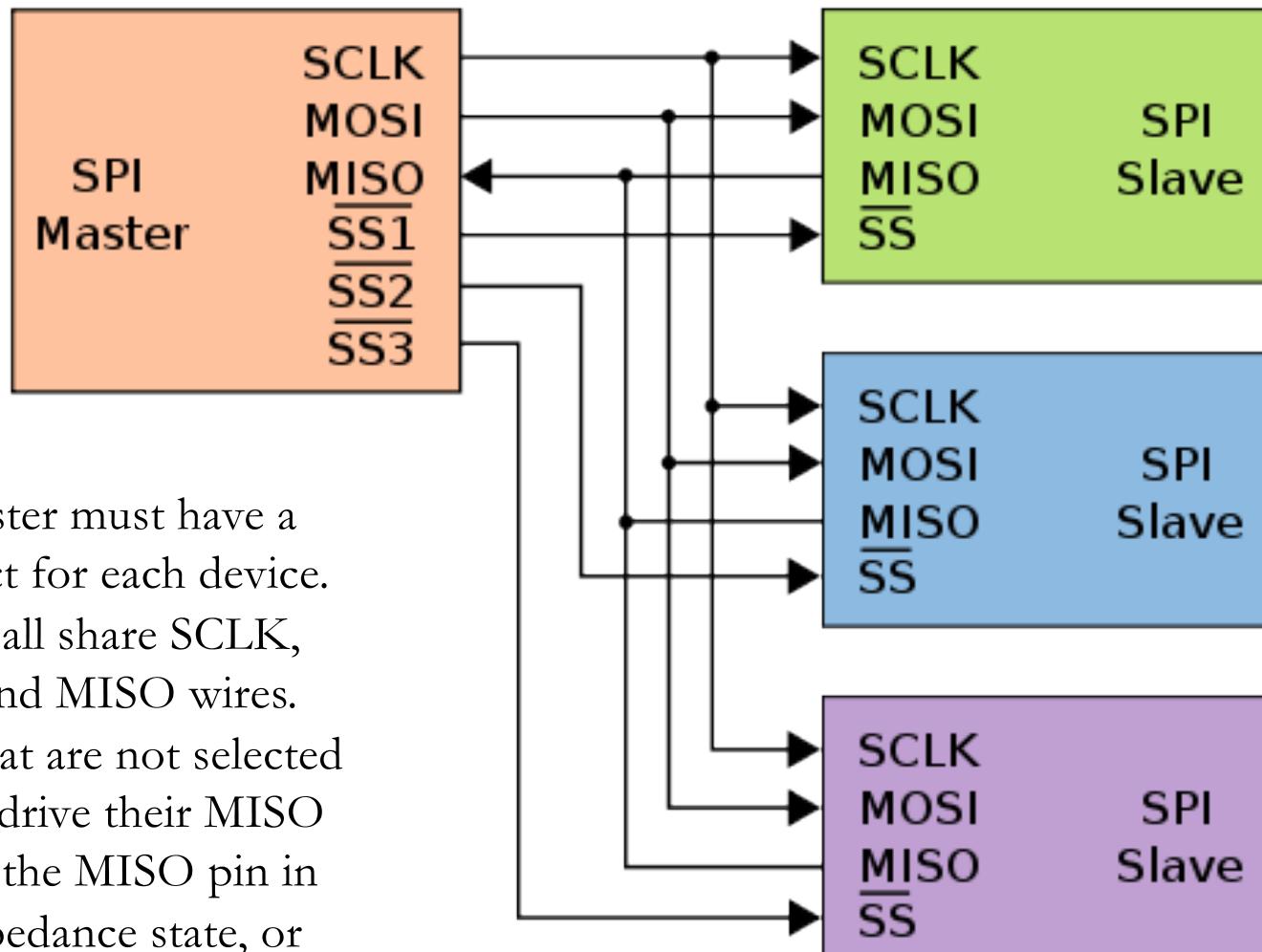


SPI Block Diagram⁽¹⁾

Atmel ATmega328P



Multiple Slave Devices



The master must have a slave select for each device.

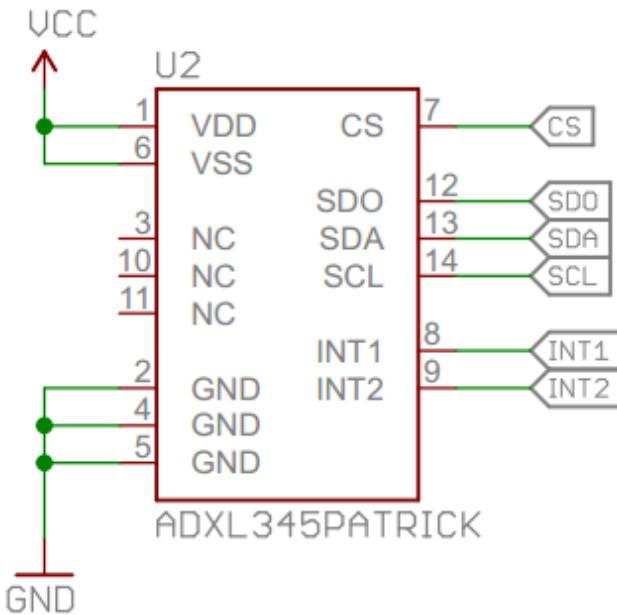
Devices all share SCLK, MOSI and MISO wires.

Devices that are not selected must not drive their MISO pins (put the MISO pin in high impedance state, or “tri-state”).

SPI Summary

- Synchronous
- Full-duplex
- Master-slave
 - Single master
 - Multiple slave
- Simple and fast
- Common sensor interface
- 4 wire
- No speed limit defined. Data rate depends on clock.
10 Mbit/s possible

Example: 3-Axis Accelerometer



FUNCTIONAL BLOCK DIAGRAM

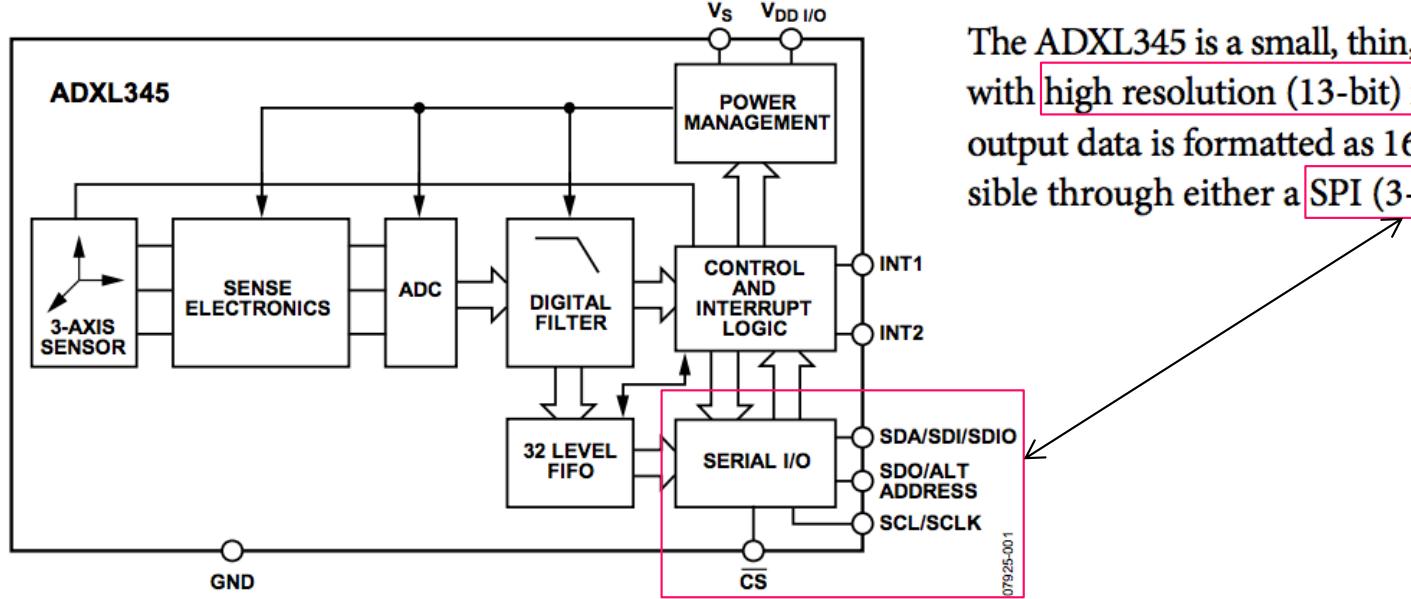
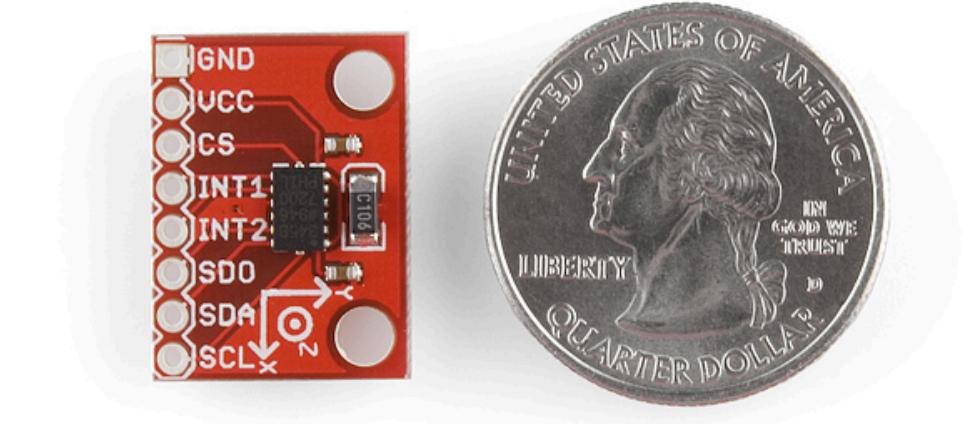


Figure 1.

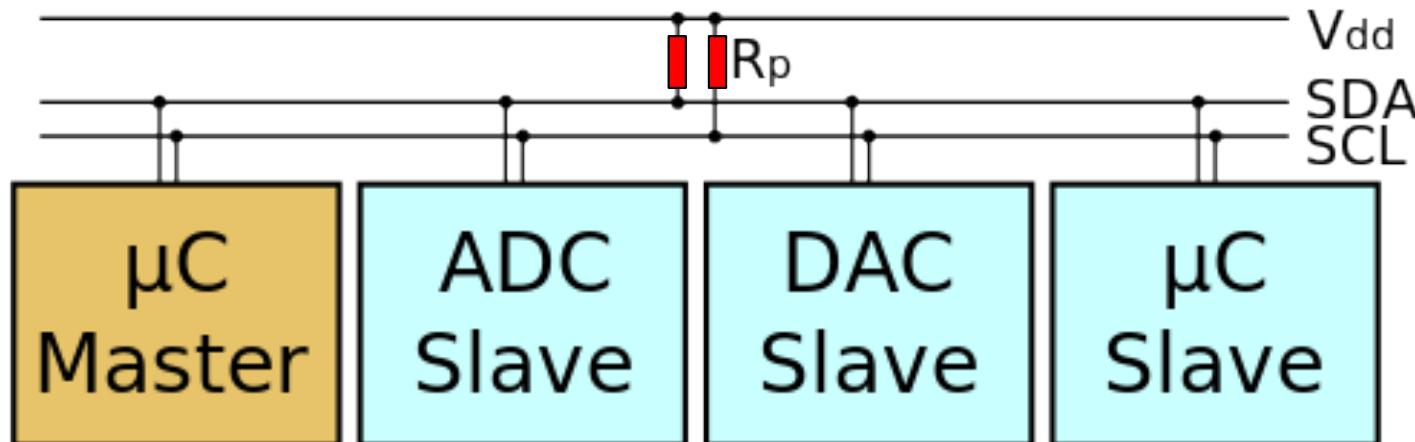


GENERAL DESCRIPTION

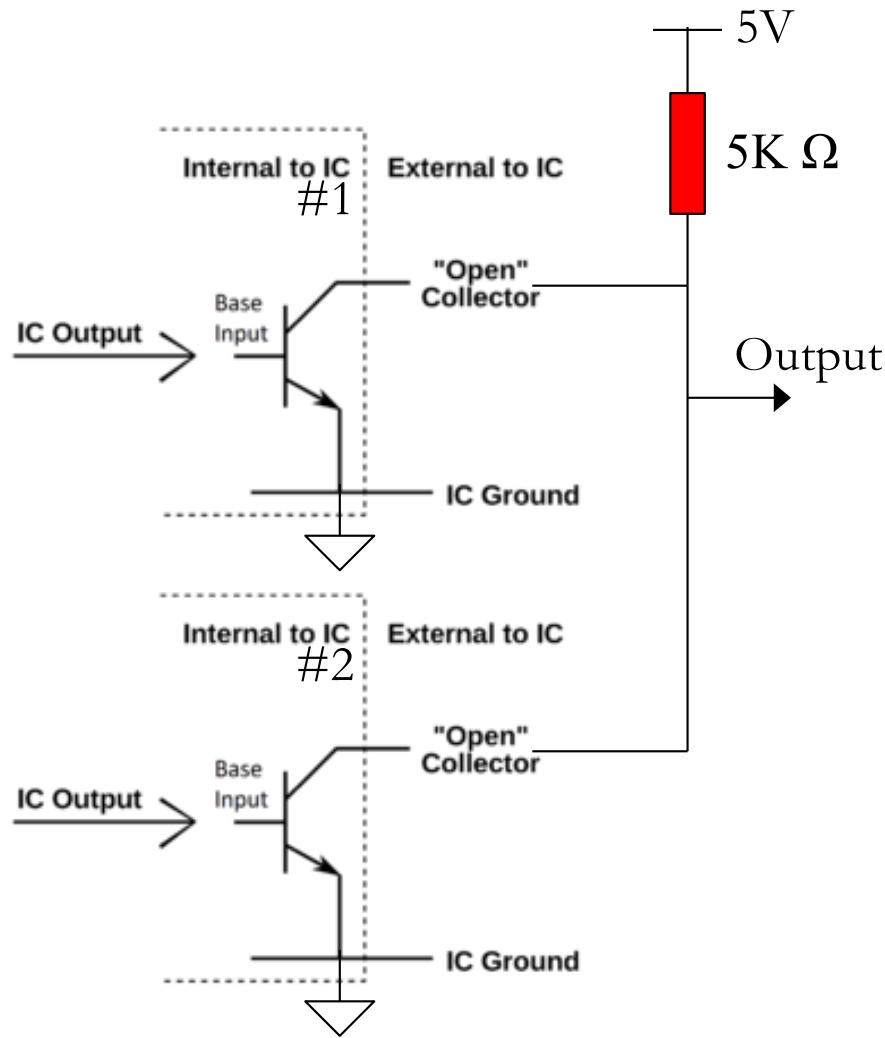
The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to $\pm 16\text{ g}$. Digital output data is formatted as 16-bit two's complement and is accessible through either a SPI (3- or 4-wire) or I²C digital interface.

I²C (Inter-Integrated Circuit)

- Uses only two bidirectional open-collector lines.
 - SDA – Serial Data Line
 - SCL – Serial Clock Line
 - Both require a pull-up resistor to Vdd (3.3V or 5V)
- Slaves are addressed.
 - 7-bit or 10-bit addressing modes
- Multi-master supported



Open-Collector Outputs



A way two ICs can “share” a wire. Either IC can transmit on the wire (but only one at a time).

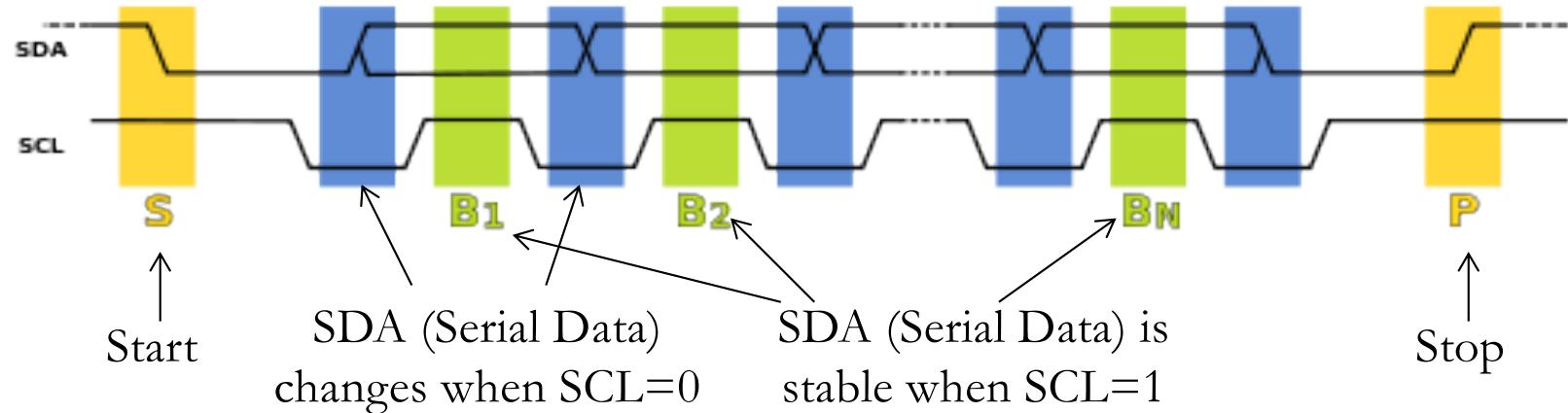
ICs can only “pull-down” (output transistor is on) or “float” (output transistor is off) the open collector wire.

If either IC “pulls-down” the output, it will go low.

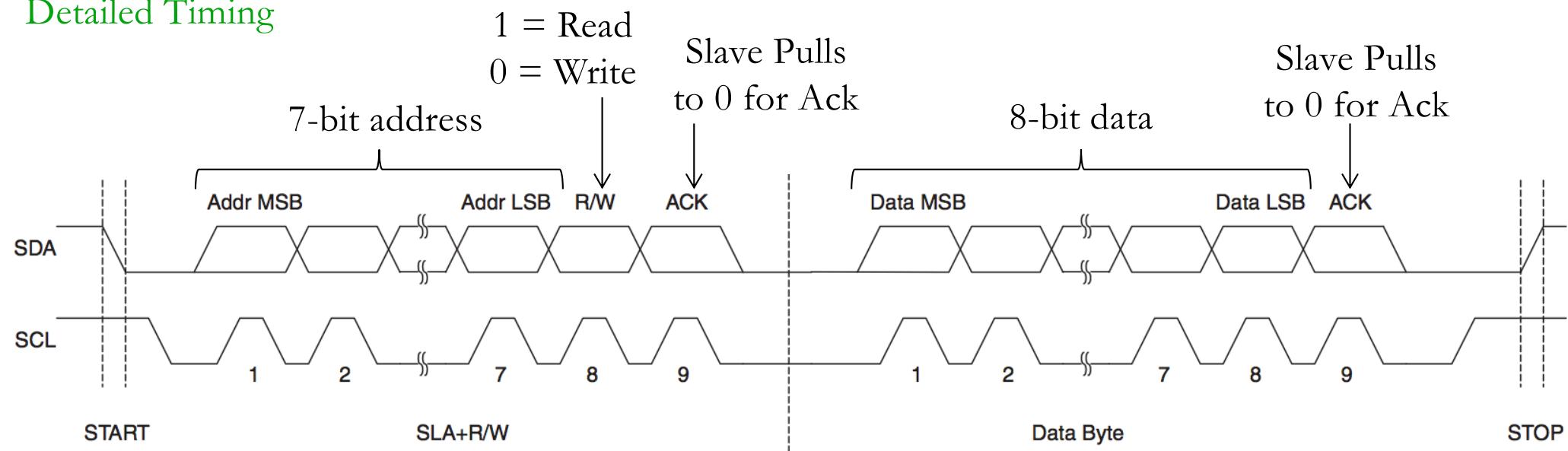
If both ICs “float” the output, the pull-up resistor will pull the output up to Vdd (+5V).

I²C Protocol

Basic Timing

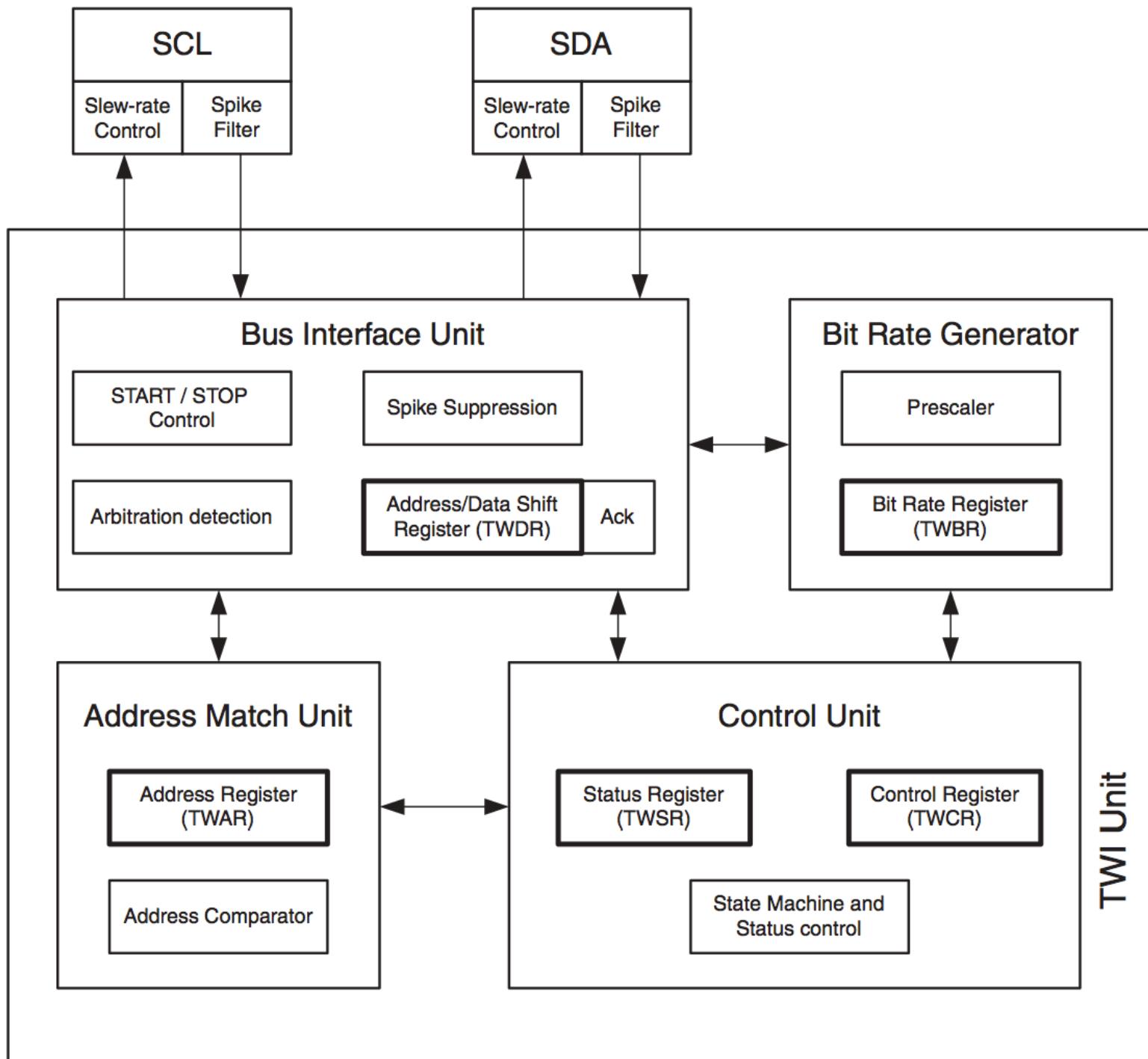


Detailed Timing



Overview of the TWI Module

Atmel ATmega328P



Other Names for I²C

- I²C was developed by Philips which later became NXP.
- Often called “Two-wire interface” or TWI to avoid trademark problems.
(NXP owns the registered trademark for I²C)
- SMBus (System Management Bus, or SMB), defined by Intel in 1995, is a subset and simpler version of the I²C protocol.
- Many sensors conform to the SMBus standard in addition to the I²C standard.

I²C / SMBus Summary

- Synchronous
- Master-Slave
 - Multiple-master
 - Multiple-slave, addressable
- Half-duplex
- Has slave acknowledge
- Slave can “stretch clock” to delay transmission
- Slower, more complex than SPI
- Common sensor interface
- 2-wire

Speeds:

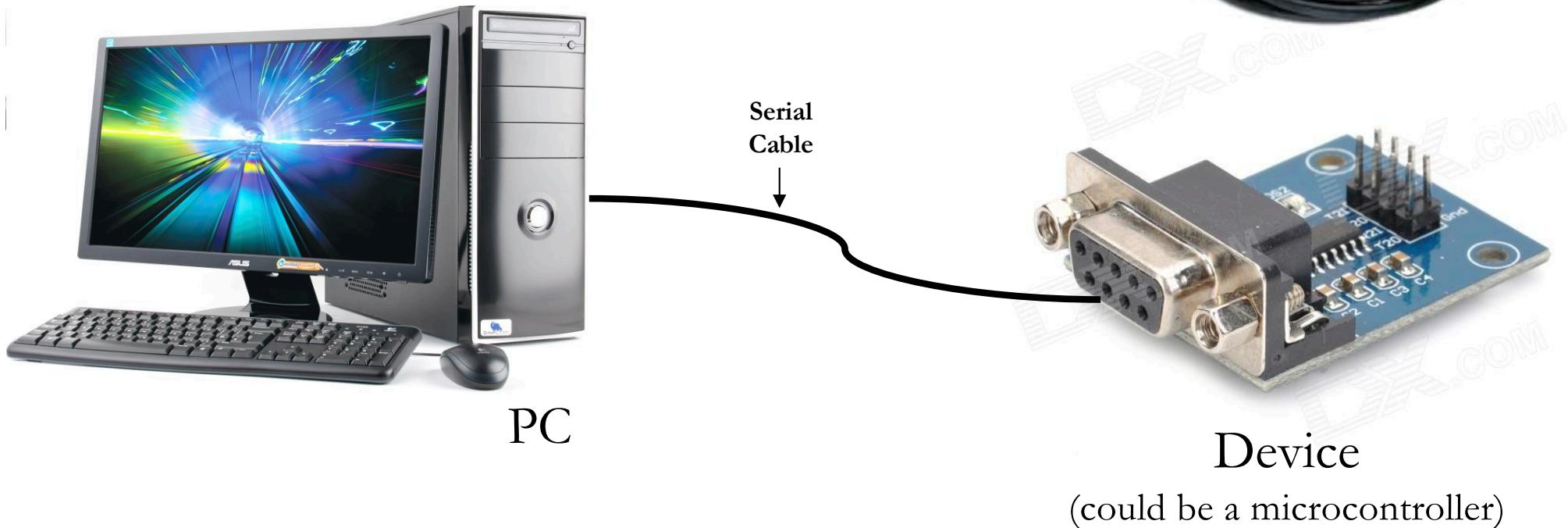
- 100 kbit/s - standard mode
- 400 kbit/s - full speed
- 1 Mbit/s - fast mode
- 3.2 Mbit/s - high speed

Lecture 25

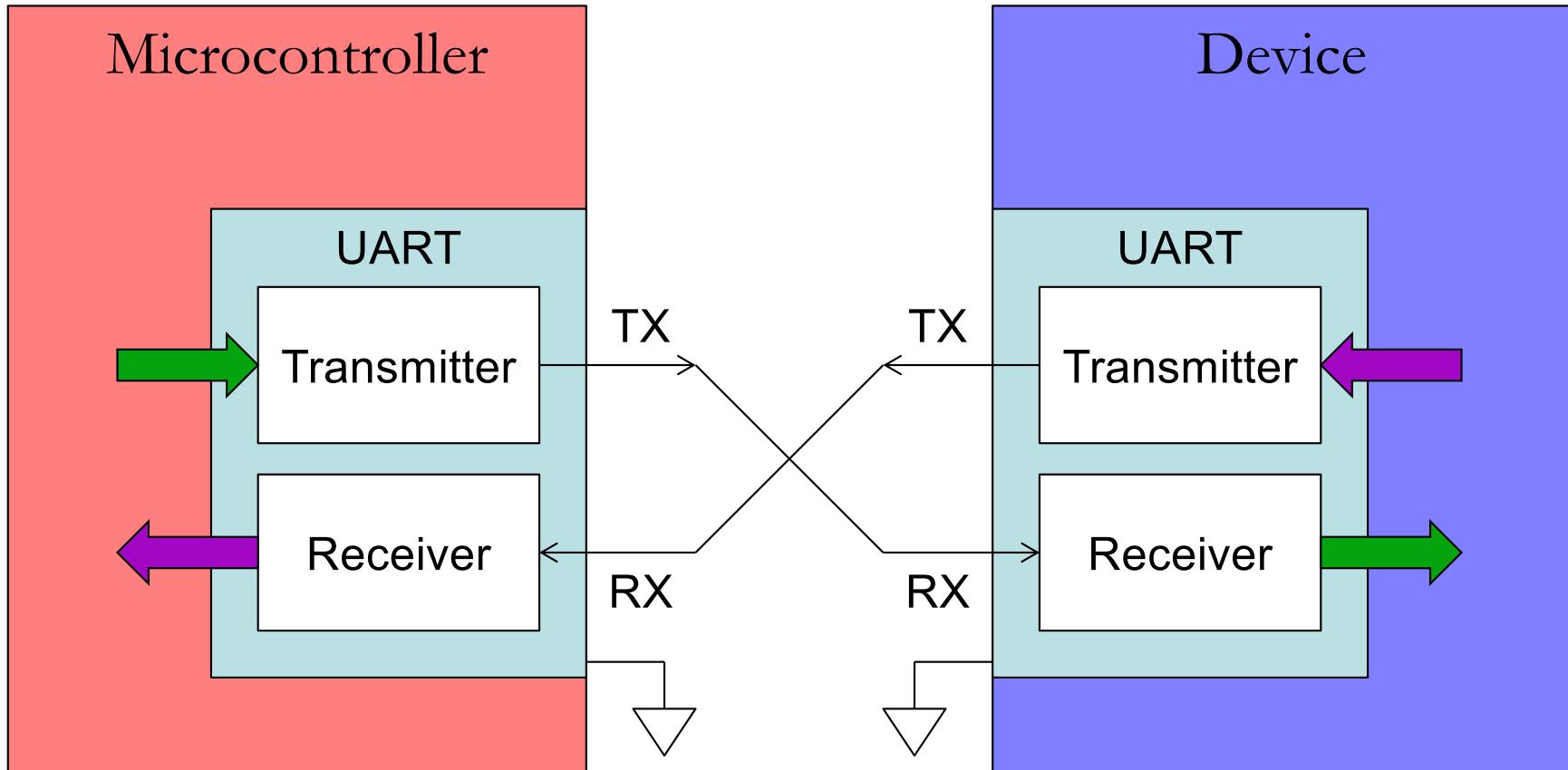
UART, USB
Interfacing to Sensors
Pulse-Width Modulation (PWM)

Uses of a UART

- PC serial port is a UART
 - In the old days, PC devices such as mice and ASCII terminals often connected to serial ports



UART Communication



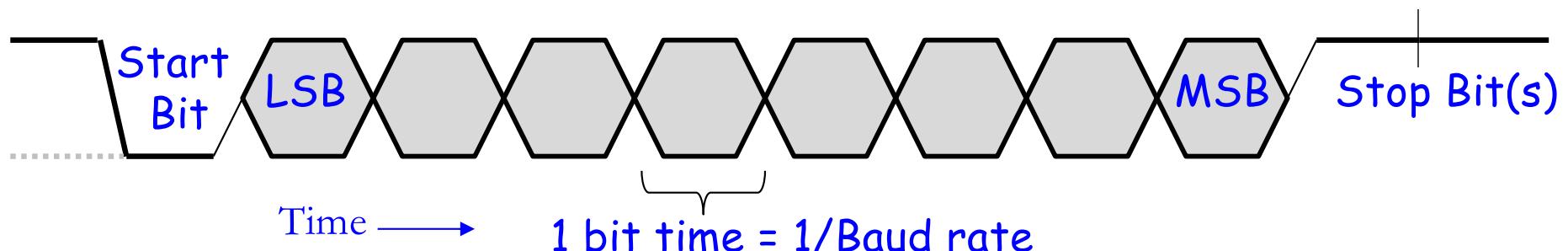
Uses a single wire for transmission in each direction (full duplex). No shared clock (asynchronous).

UART

(Universal Asynchronous Receiver/Transmitter)

- Oldest and slowest of the serial protocols
- Point-to-point (only links two devices together)
- Transmits data from one device to another by sending a start bit followed by sending data one bit at a time starting with the least significant bit (LSB) first.

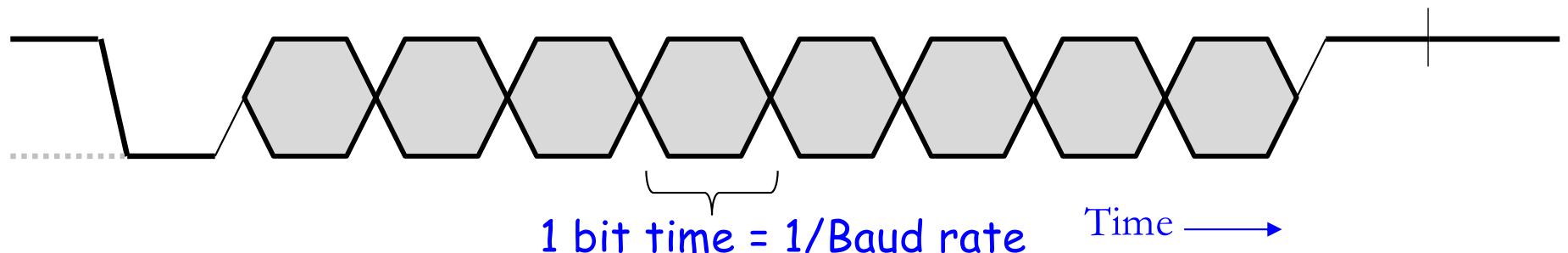
Bit number	1	2	3	4	5	6	7	8	9	10	11
	5–8 data bits								Stop bit(s)		
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Stop	



UART Character Transmission

- **Asynchronous.** The transmitter does not send its clock to the receiver. The receiver must generate its own clock.
- Each bit is transmitted for a fixed bit-time determined by the transmission rate called the **Baud Rate**.
- Transmitter and Receiver must both be configured with the exact same baud rate to communicate.

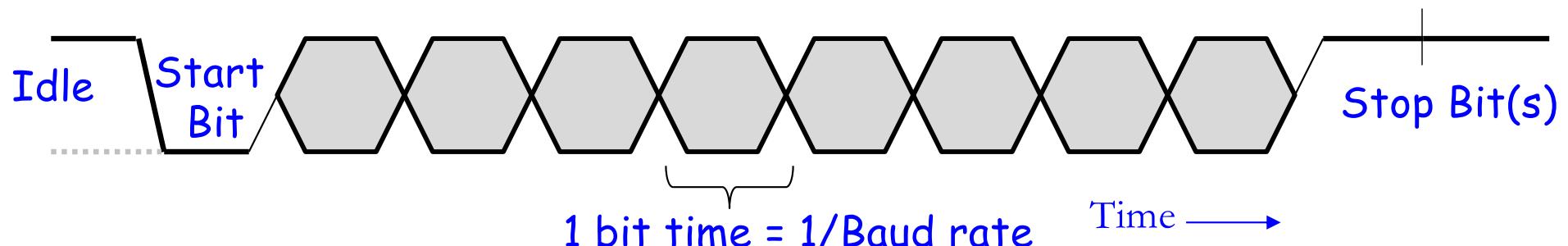
Bit number	1	2	3	4	5	6	7	8	9	10	11
	Start bit	5–8 data bits									Stop bit(s)
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Stop	



UART Character Transmission

- Transmission lines idle at a high voltage.
- The **start bit** marks the beginning of a new transmission, the receiver synchronizes its clock with the falling edge of the start bit.
- Transmitter and Receiver must both be configured with the exact same **baud rate** to communicate.

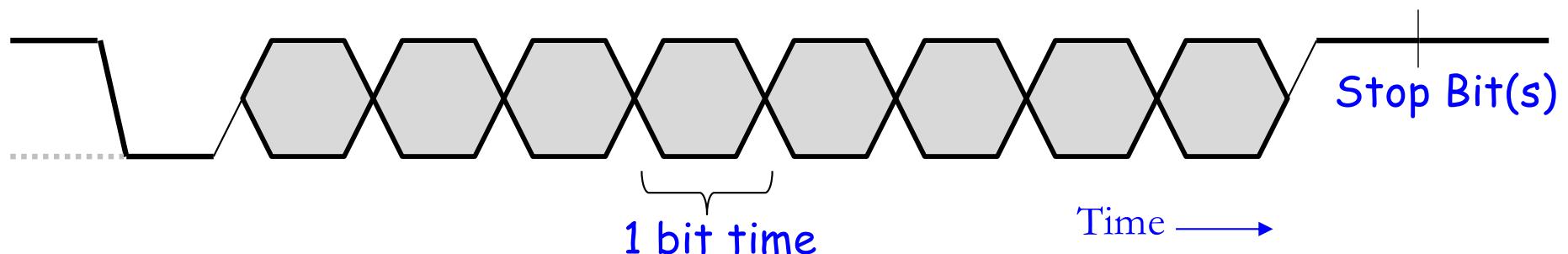
Bit number	1	2	3	4	5	6	7	8	9	10	11
	Start bit		5–8 data bits								
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Stop	



UART Character Transmission

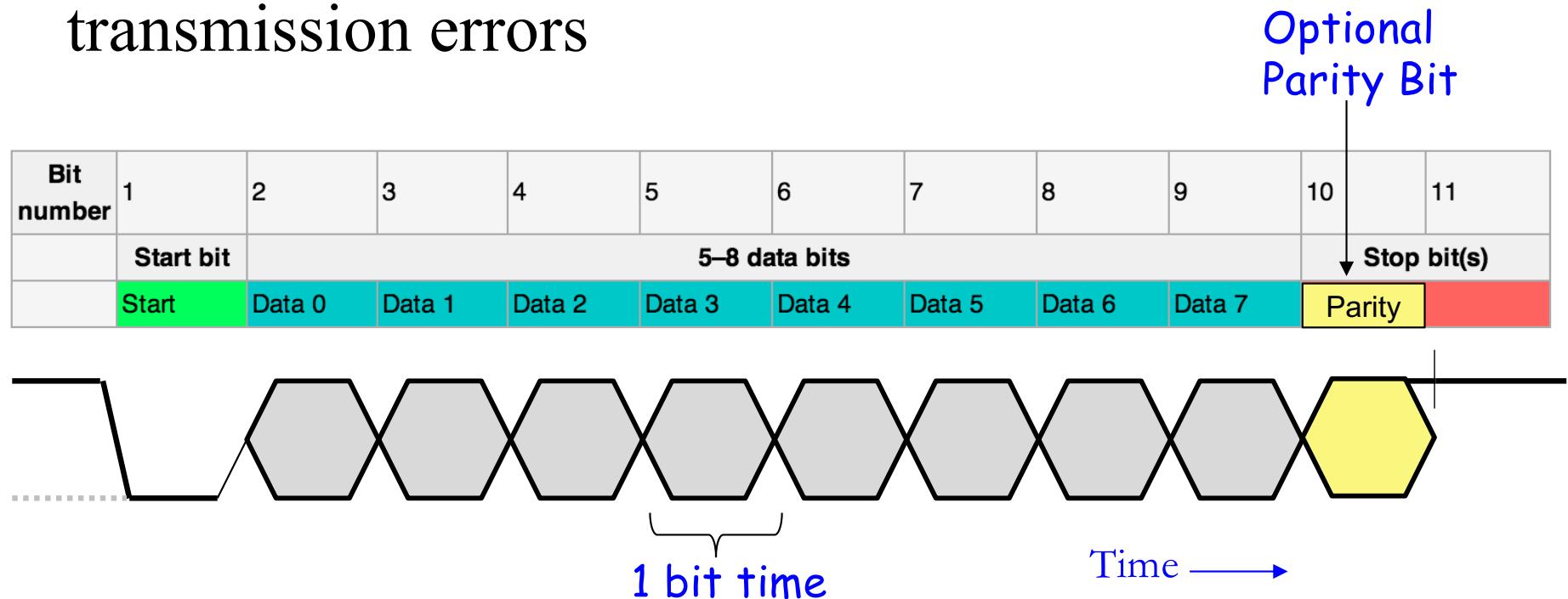
- One, one and a half, or two **stop bits**
- The **stop bit(s)** marks the end of transmission
- Receiver checks to make sure it is ‘1’ (mark)
- Separates one word from the start bit of the next word

Bit number	1	2	3	4	5	6	7	8	9	10	11
	Start bit	5–8 data bits									Stop bit(s)
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Stop	



UART Character Transmission

- An optional **parity bit** is added to make the number of 1's in the (data + parity) bits either
 - even (even parity), or odd (odd parity)
- This bit can be used by the receiver to check for transmission errors



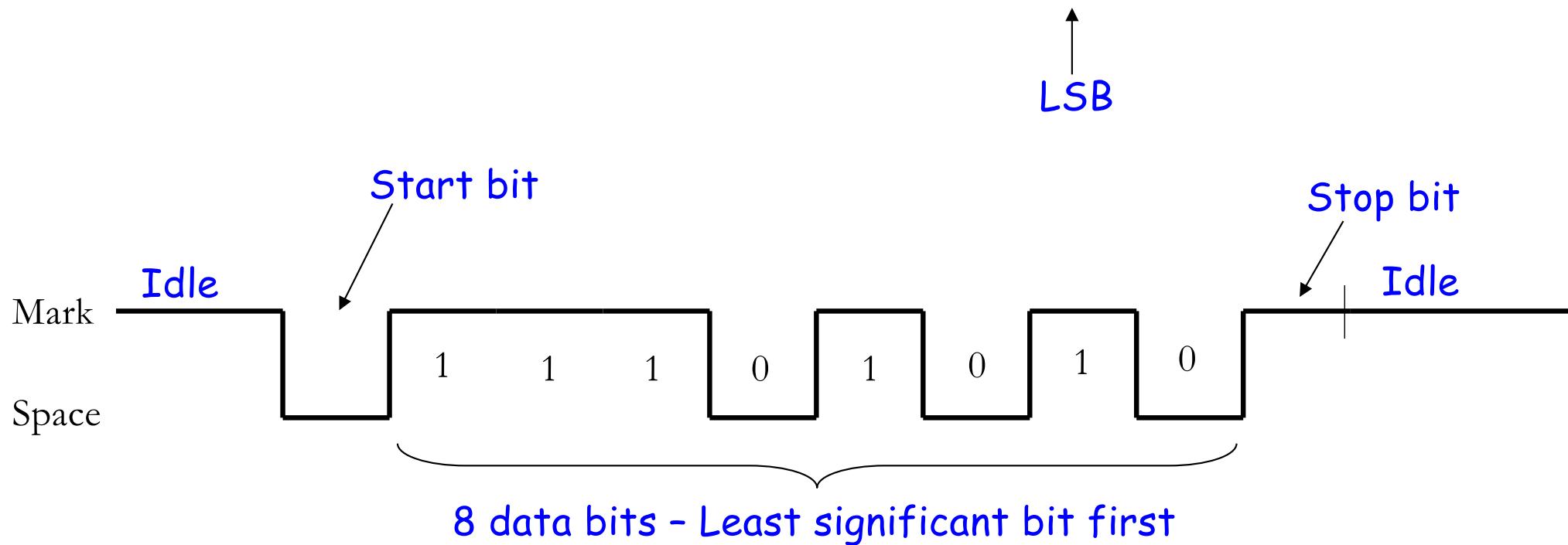
UART Configuration Options

- Serial Port Configuration
 - **Baud Rate** = Bits per second (9600, 19200, 38400, ...)
 - **Number of data bits** = 7, 8
 - **Stop bits** = 1, 1.5, 2
 - **Parity** = None, Even, Odd
- Most common configuration
 - **Baud Rate** = 9600
 - **Number of data bits** = 8
 - **Stop bits** = 1
 - **Parity** = None



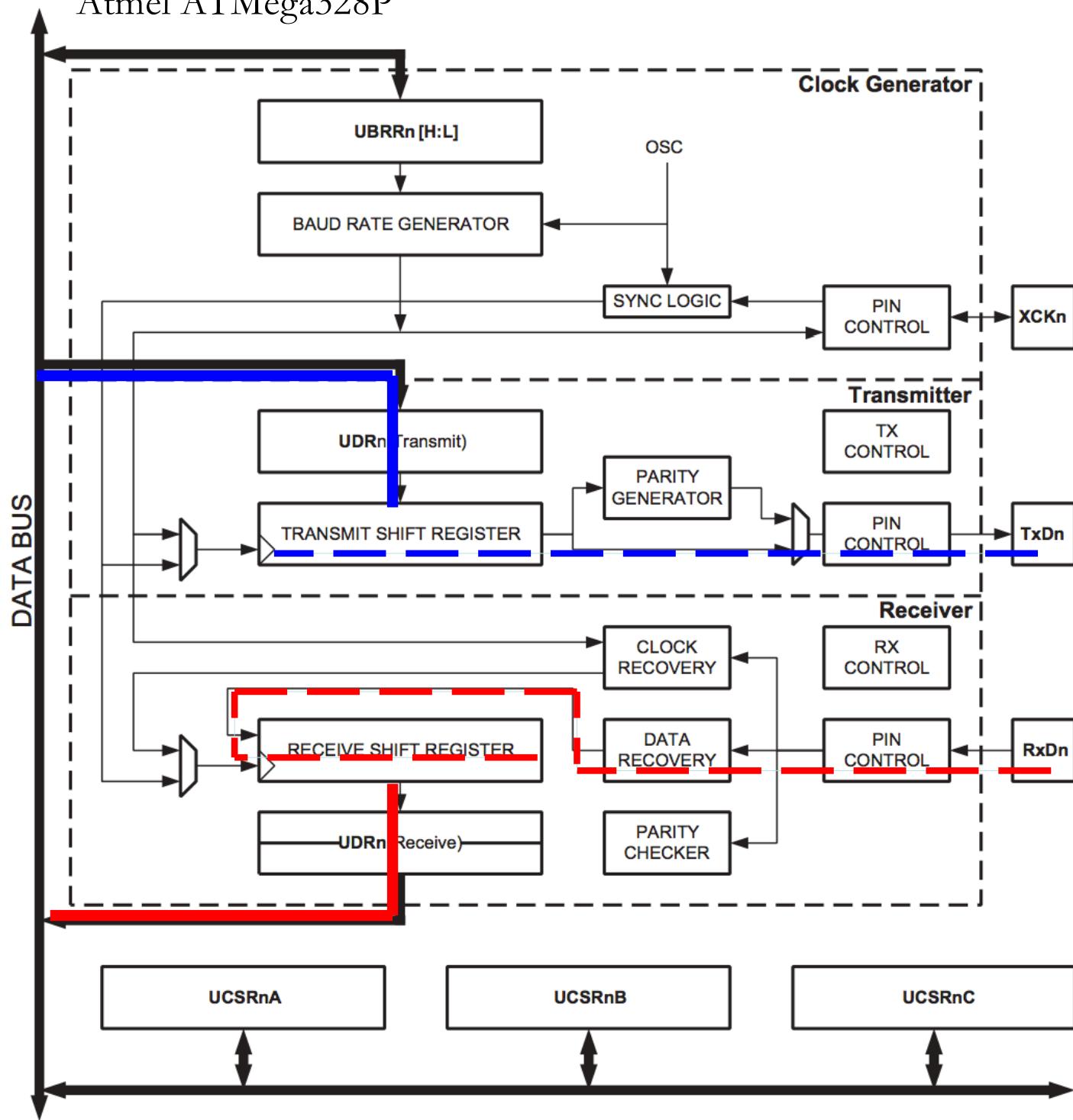
UART Transmission Example

- Send the ASCII letter ‘W’ (01010111)



USART Block Diagram⁽¹⁾

Atmel ATmega328P

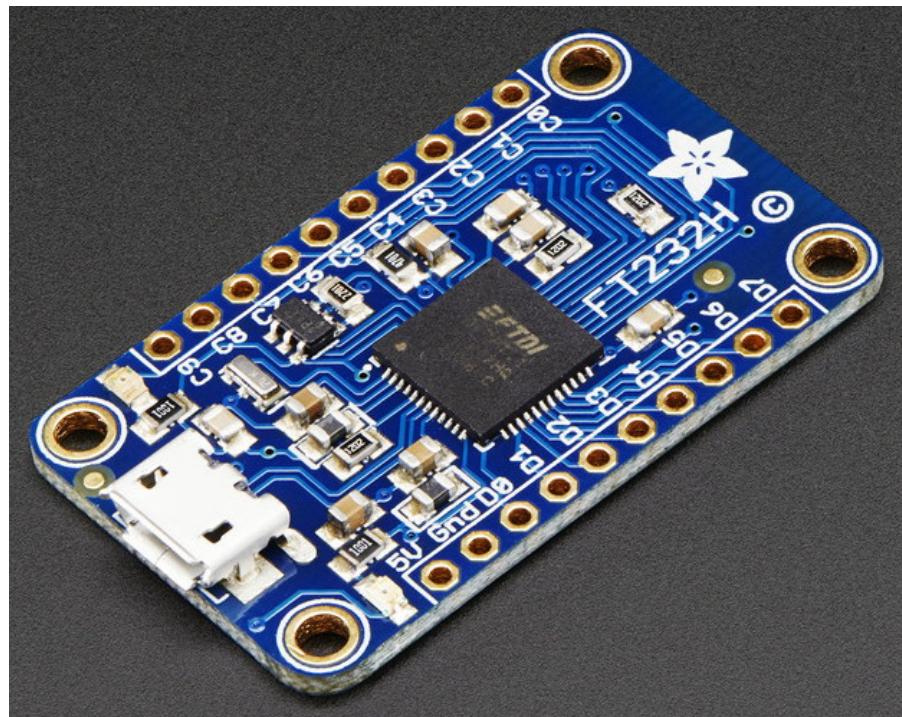


UART Summary

- Oldest of the serial communication protocols
- Point-to-point (only links two devices)
- Full-duplex
- Asynchronous (baud rates must match)
- 2-wire
- Only transmits 8 bits at a time
- Not very fast
- Pretty much replaced with USB for PCs, but still find on some equipment

UART Example

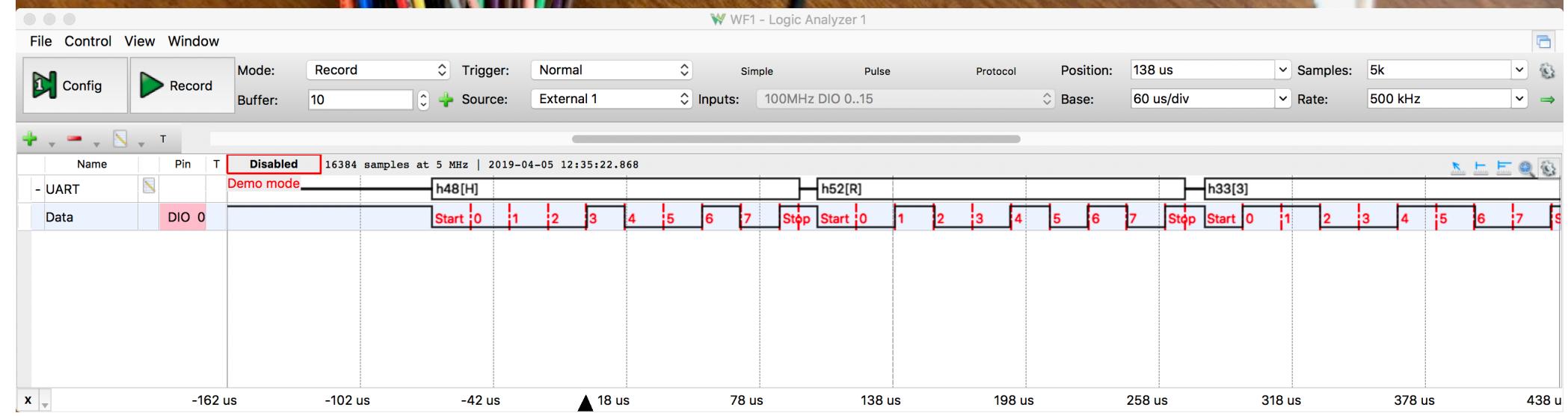
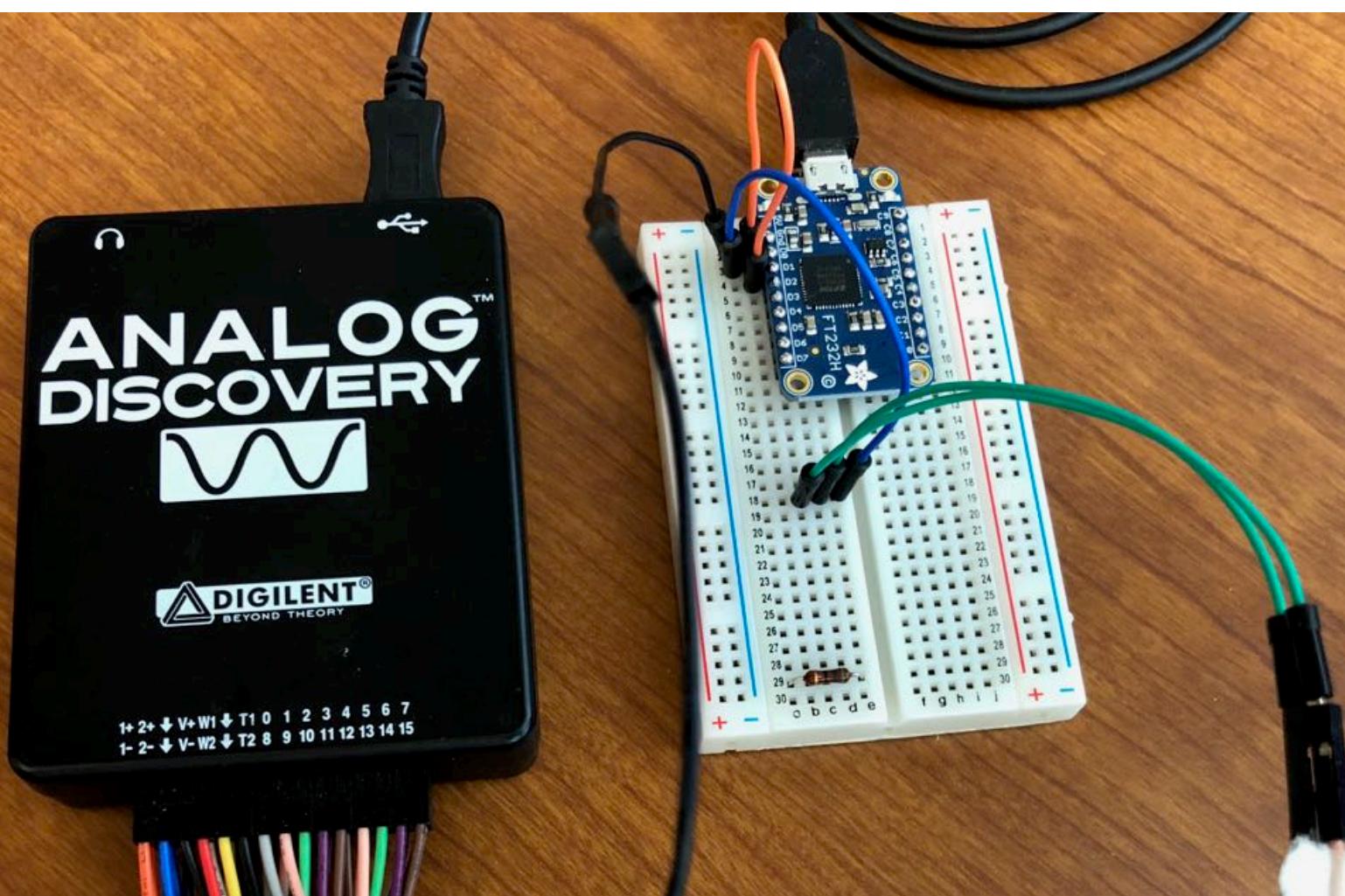
- Need to connect mac laptop directly to motor controller that has serial UART



Adafruit FT232H
Breakout - General
Purpose USB to
GPIO+SPI+I2C

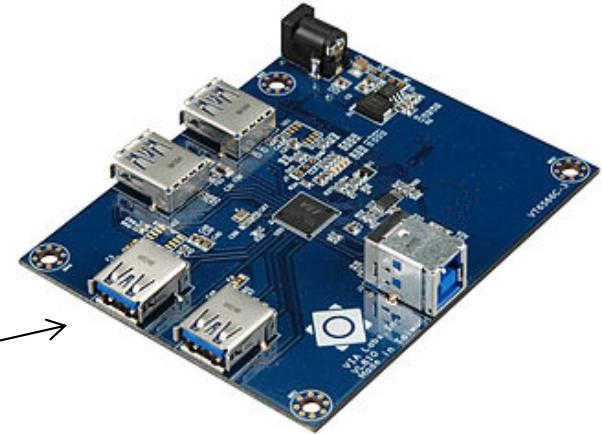
PRODUCT ID: 2264

\$14.95



USB Summary

- Serial, Differential Data Signaling
- Master (host) – slave (peripheral)
- Can connect to multiple peripherals using hub →
- Half-duplex
- Very complex protocol
- Fast
- Can supply power (+5V, 500 mA max) to peripherals devices



Pin 1 V_{CC} (+5 V, red wire)
Pin 2 Data– (white wire)
Pin 3 Data+ (green wire)
Pin 4 Ground (black wire)



USB

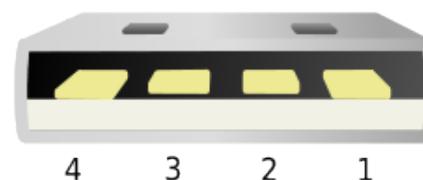
Standard B

+ D–

1 2

Standard A

– D+ D– +



1996	USB 1.1	12 Mbits/sec
2000	USB 2.0	480 Mbits/sec
2008	USB 3.0	5.0 Gbits/sec
2013	USB 3.1	10.0 Gbits/sec
2017	USB 3.2	20.0 Gbits/sec



Mini-USB

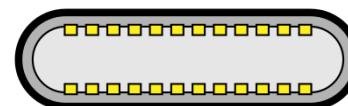


Micro-USB

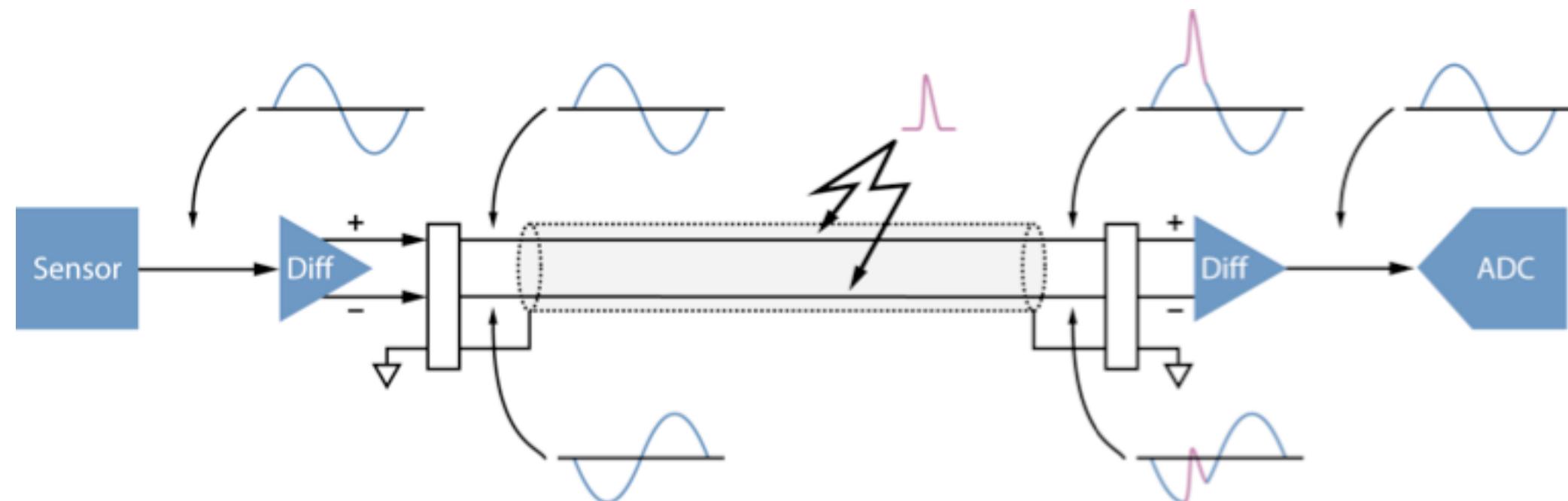
USB-C



24 pin
Up to 20V 5A



Differential Data Transmission



Differential data transmissions – fast and noise tolerant

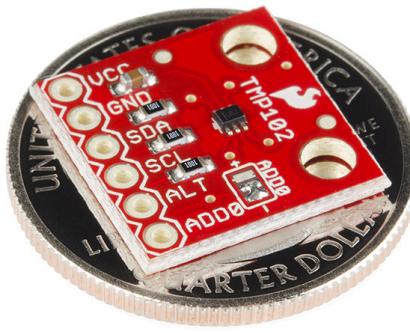
Serial Communication Comparison

Serial Comm Type	Sync - Asynch	Architecture	Half or Full Duplex	Addres sed?	Proto- col	Wires
SPI	Synch- ronous	Master- Slave	Full Duplex	No	Simple	4
I ² C	Synch- ronous	Master- Slave	Half Duplex	Yes	Complex	2
UART	Asynch- ronous	Point-to- Point	Full Duplex	No	Simple	2
USB 2.0	Asynch- ronous	Virtual Endpoin t	Half Duplex	No	Very Complex	2 + 2 (with Power)

Sensors

- A **sensor** is a converter that measures a physical quantity and converts it into a signal which can be read by an electric instrument.

Temperature Sensor



DESCRIPTION

The TMP102 is a **two-wire, serial** output temperature sensor available in a tiny SOT563 package. Requiring no external components, the TMP102 is capable of reading temperatures to a resolution of 0.0625°C .

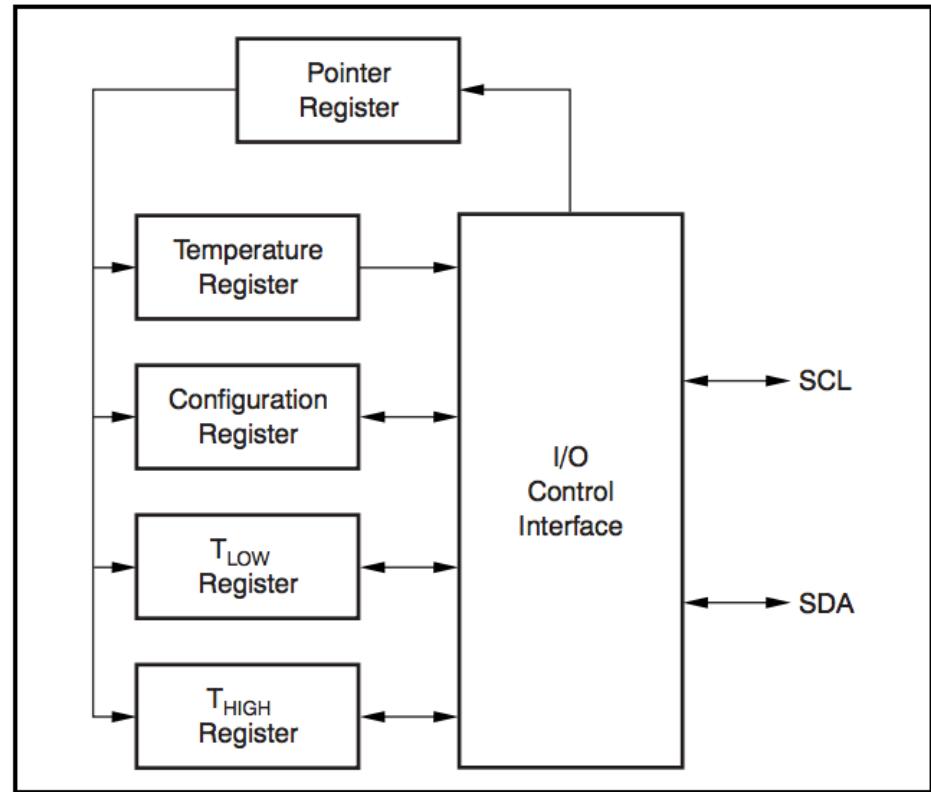
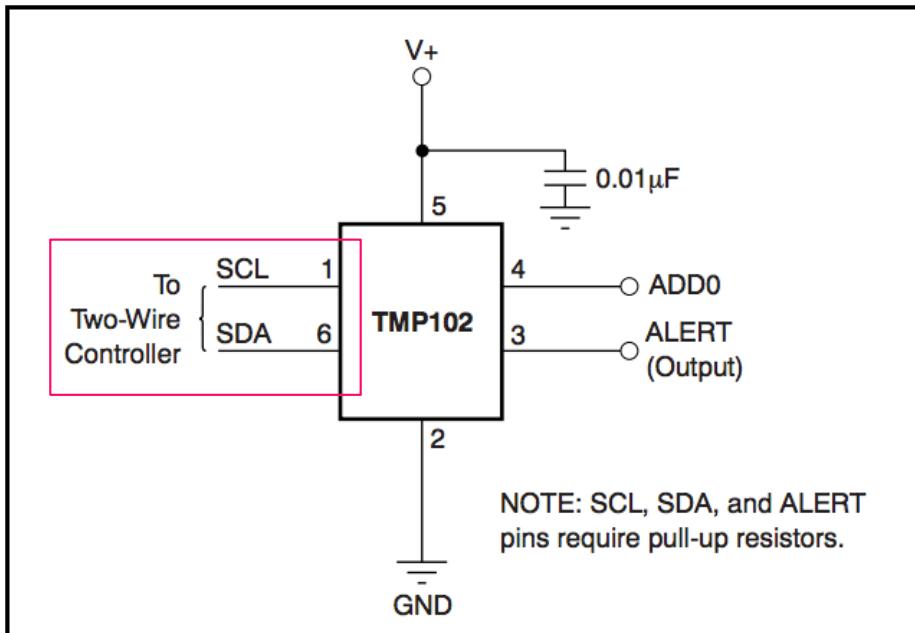


Figure 8. Internal Register Structure

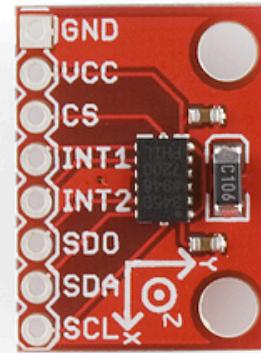
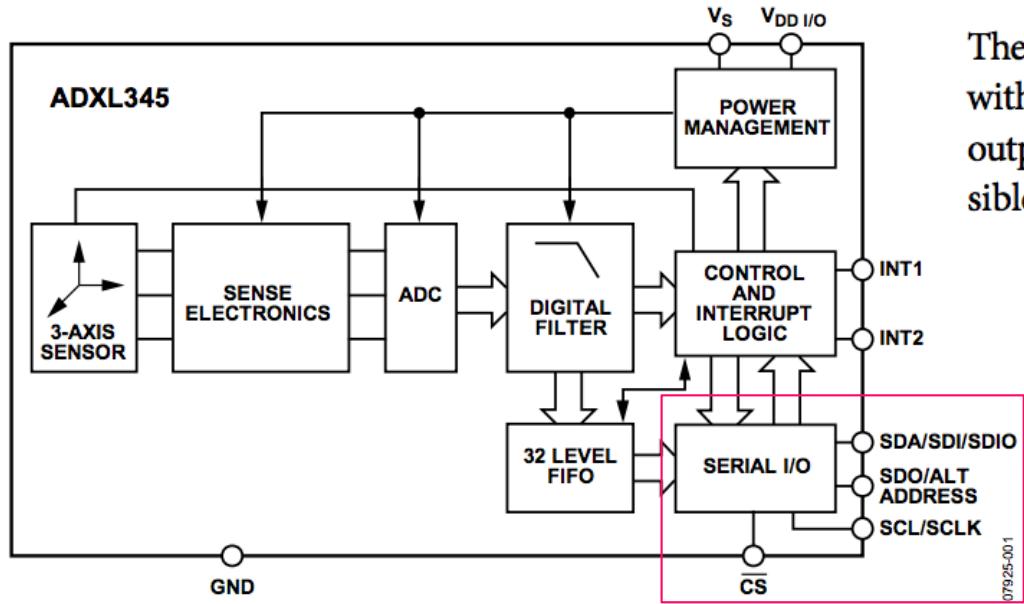
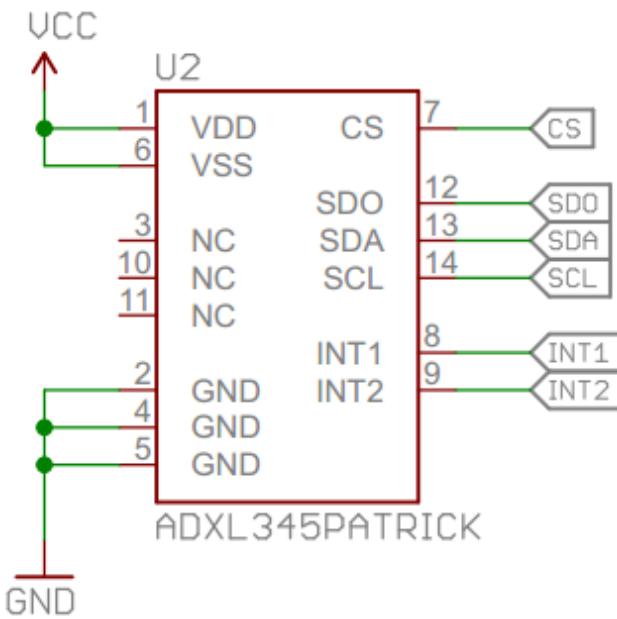
Table 1. Pointer Register Byte

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	0	Register Bits

Table 2. Pointer Addresses

P1	P0	REGISTER
0	0	Temperature Register (Read Only)
0	1	Configuration Register (Read/Write)
1	0	T_low Register (Read/Write)
1	1	T_high Register (Read/Write)

3-Axis Accelerometer

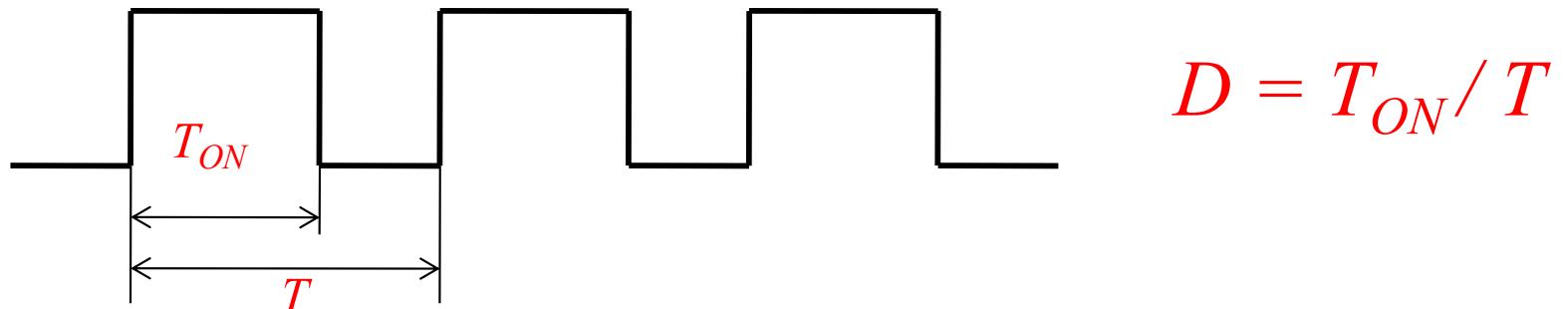


GENERAL DESCRIPTION

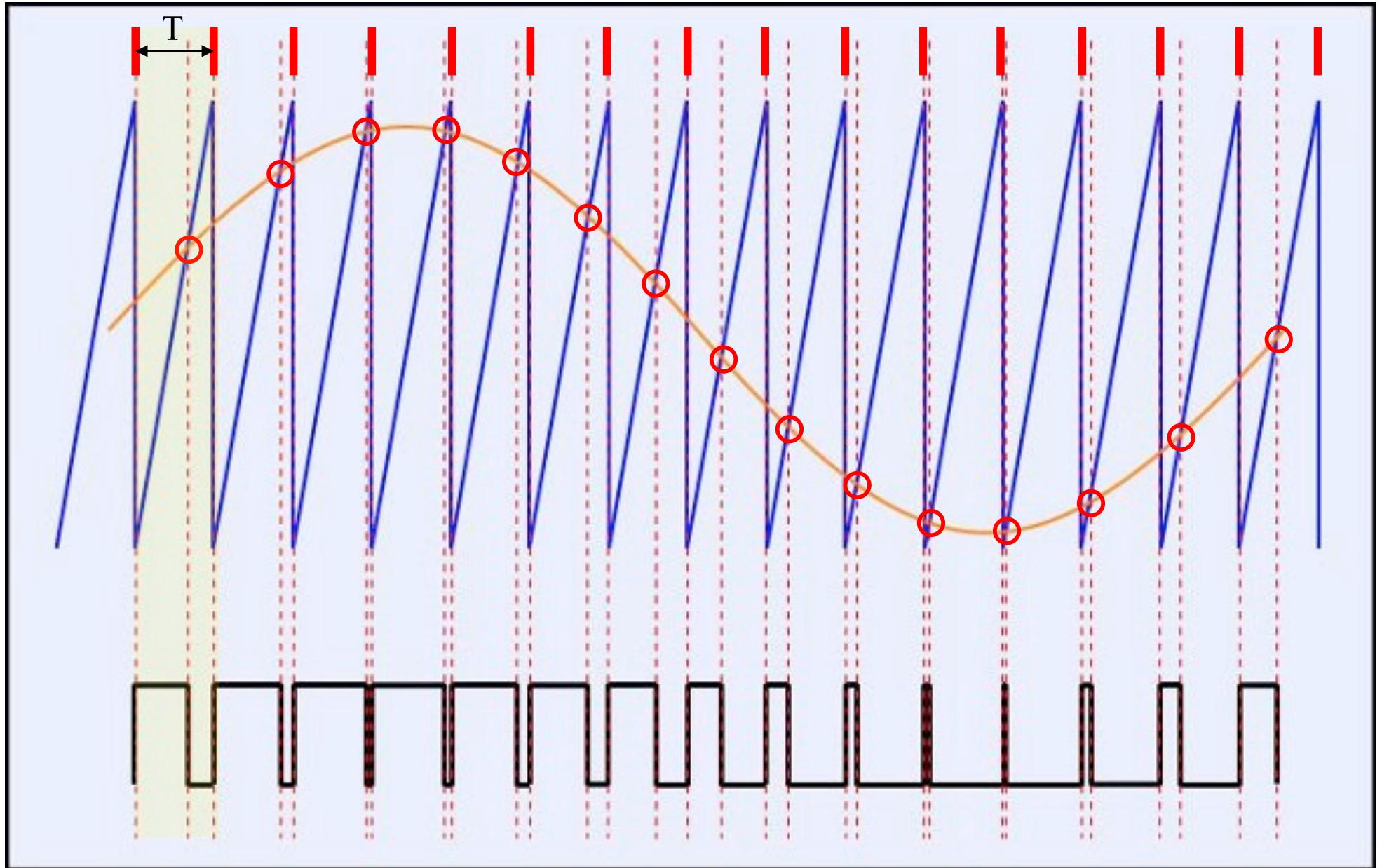
The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to $\pm 16\text{ g}$. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I²C digital interface.

Pulse Width Modulation

- A method of regulating the amount of voltage/power delivered to a load.
- The **average value** of the voltage fed to the load is controlled by switching between a low and high voltage at a high **switching frequency** $f_{sw} = 1/T$.
- The **duty cycle D** is the ratio of 'on' time to the switching period $T = 1/f_{sw}$.

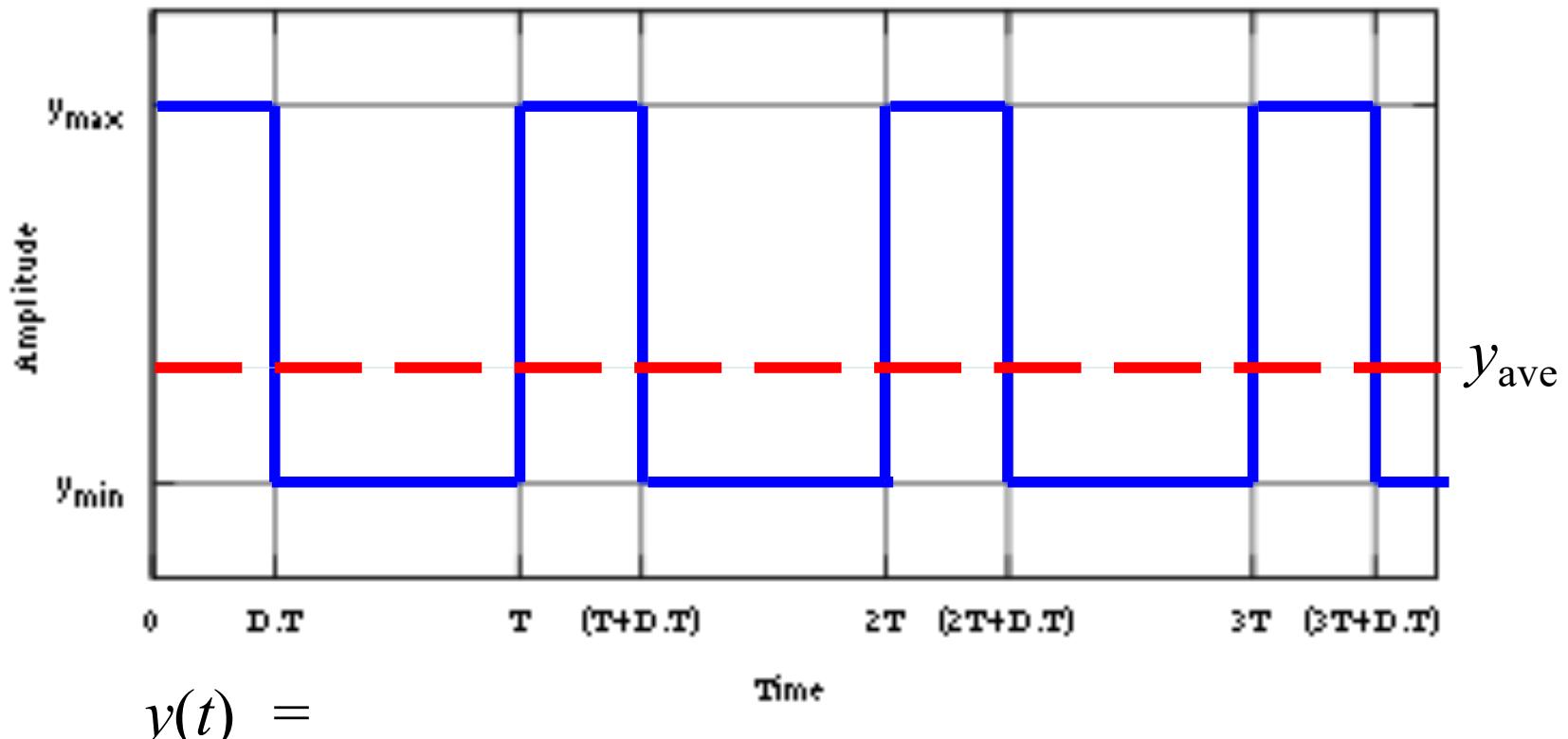


Pulse Width Modulation



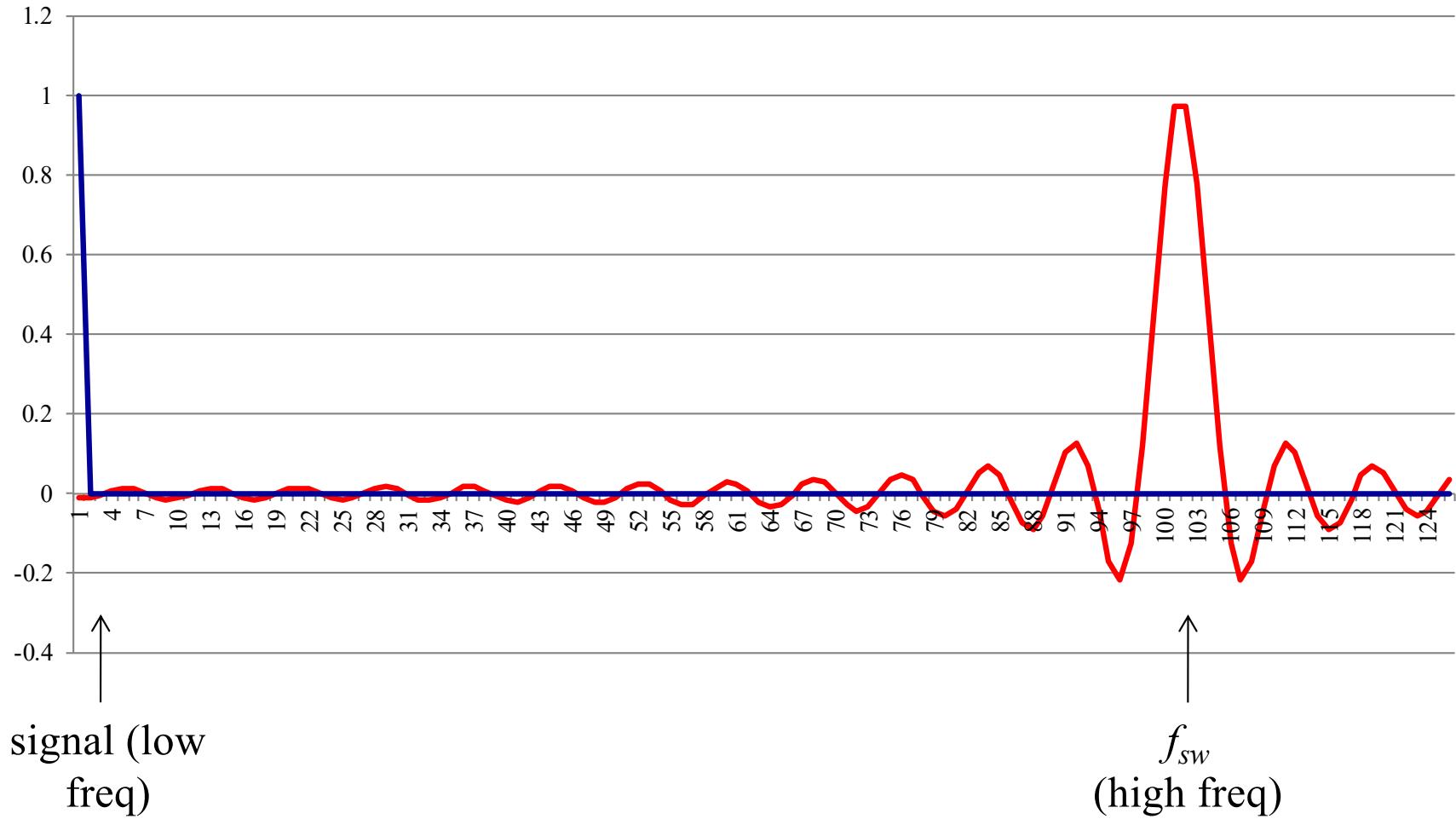
Fixed PWM Switching Period/Frequency

PWM Average Value

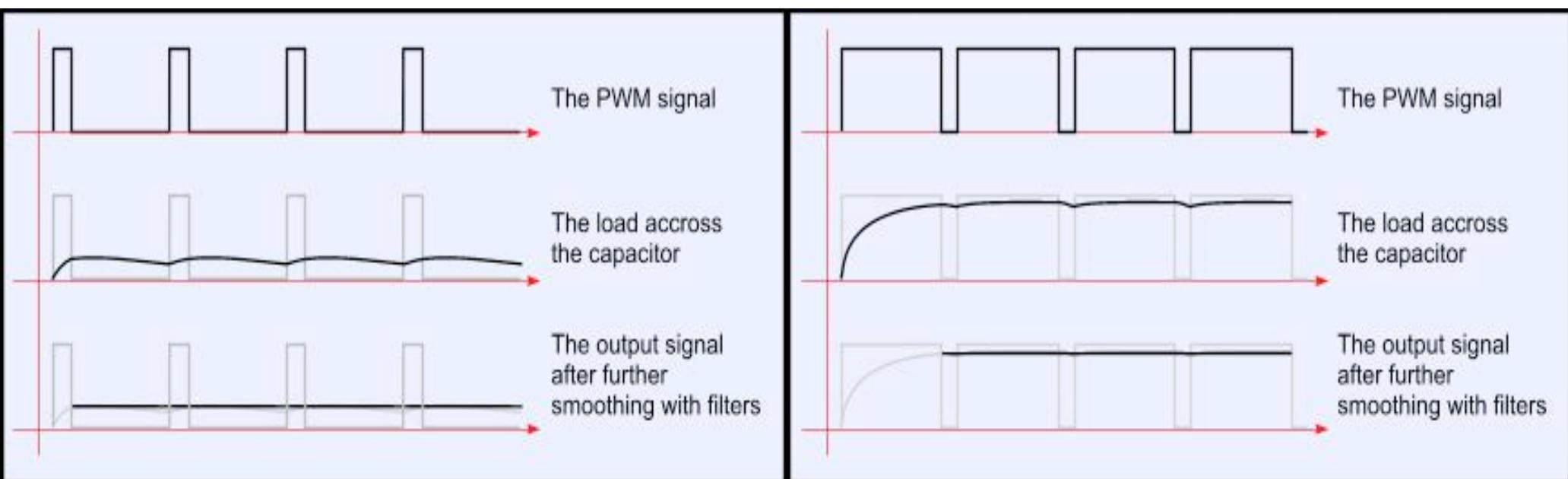
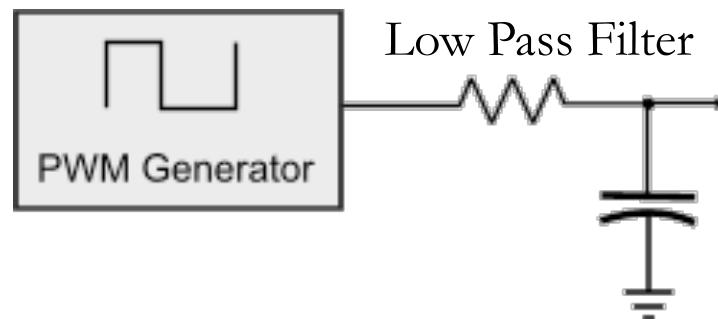


$$\begin{aligned} y_{ave} &= D y_{max} + (1 - D) y_{min} \\ &= D y_{max} \quad (\text{if } y_{min} = 0) \end{aligned}$$

Frequency Power Spectrum



Filtering to Get the Output

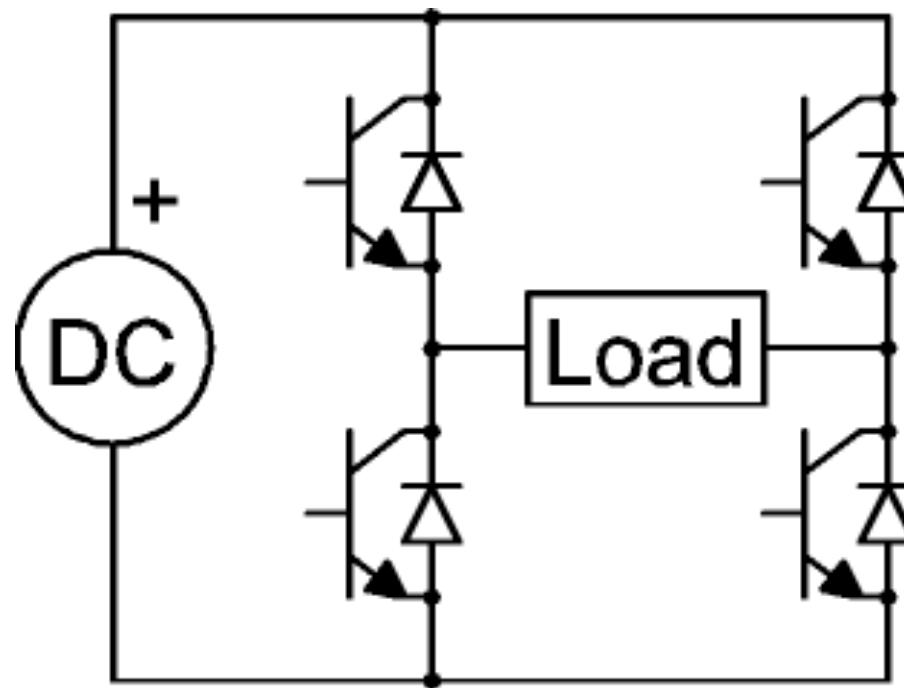


Some Applications of PWM

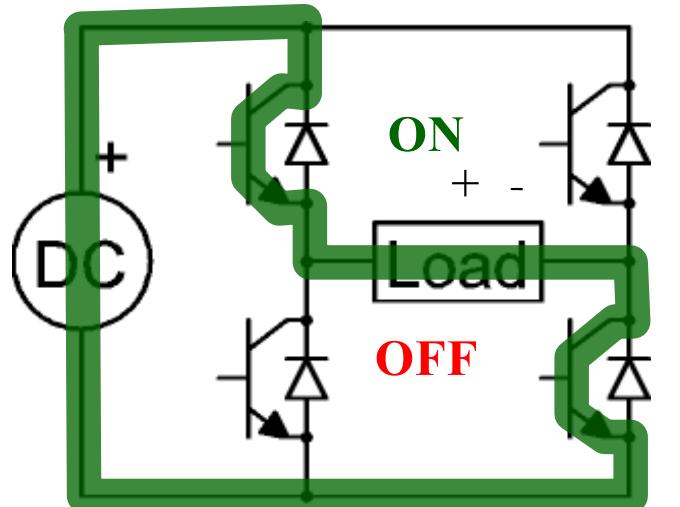
- Modulate, transmit and/or store **analog signals** like audio/voice telecommunications and music.
- The output power and/or voltage of **switching power supplies** can be controlled digitally.
- The torque and speed of a DC **motor** can be controlled by altering the voltage or the duty cycle of the PWM.
- PWM is widely used in **dimmer circuits** that could control a variety of lamps including LEDs.

H-Bridge

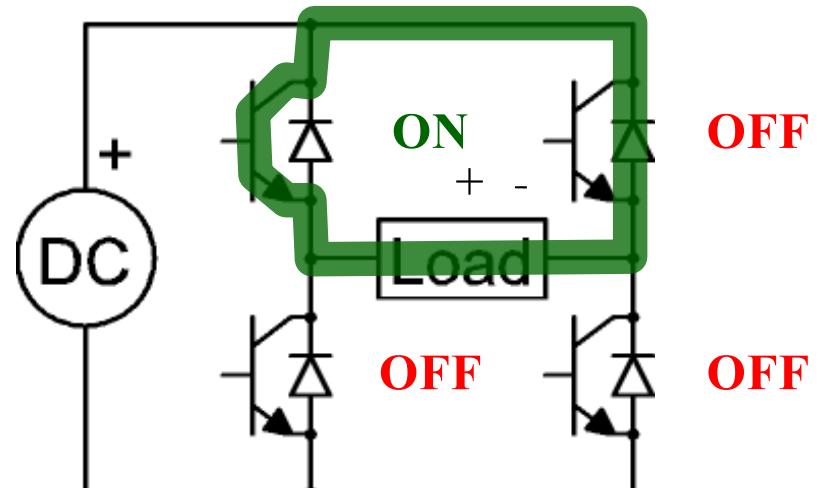
- Circuit that enables a voltage to be applied across a load in either direction.
 - Used to implement an **Inverter** circuit that converts DC to AC.
 - Used to control DC **motors** in both forward and reverse



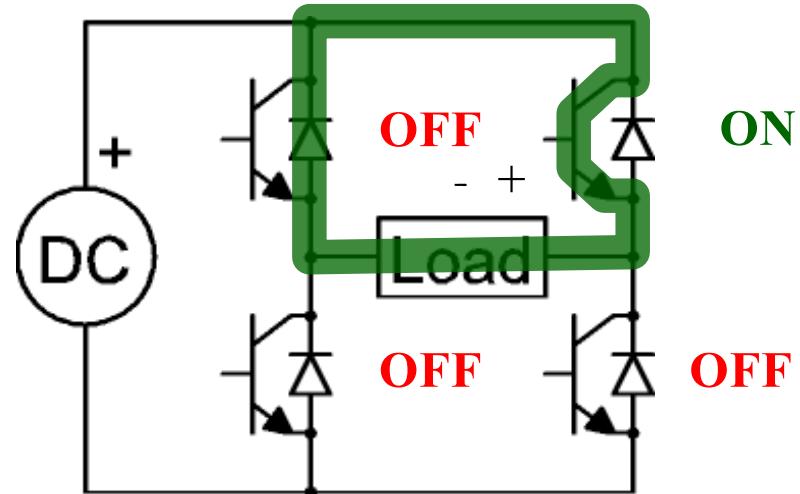
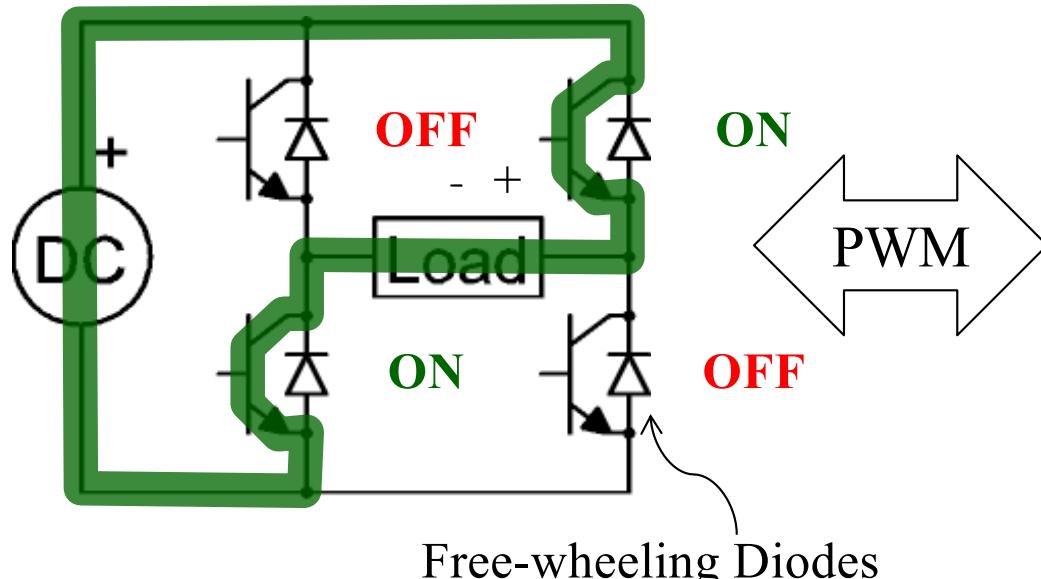
Operational States of Inverter



OFF
ON
PWM

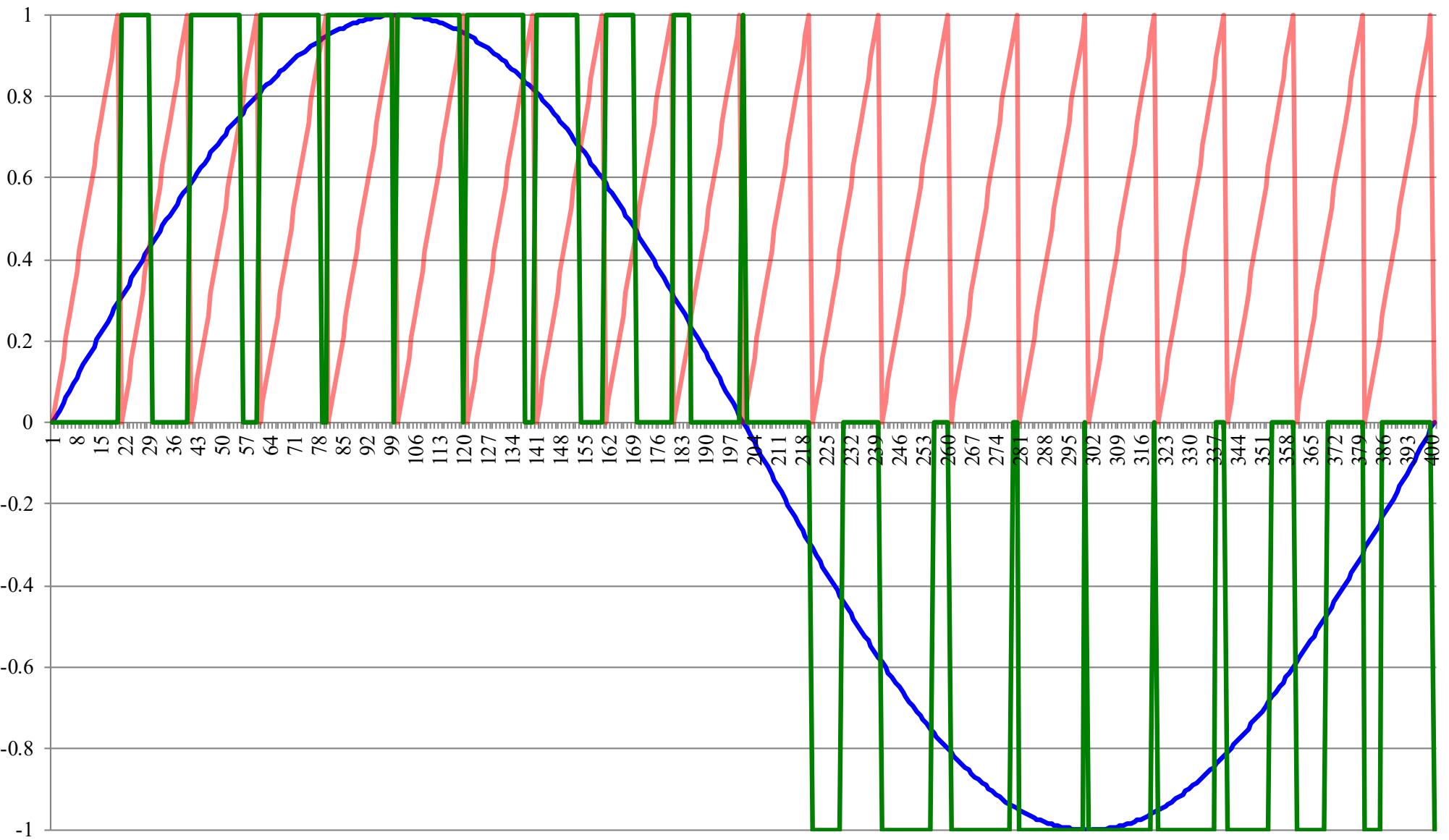


OFF
OFF
Change Direction

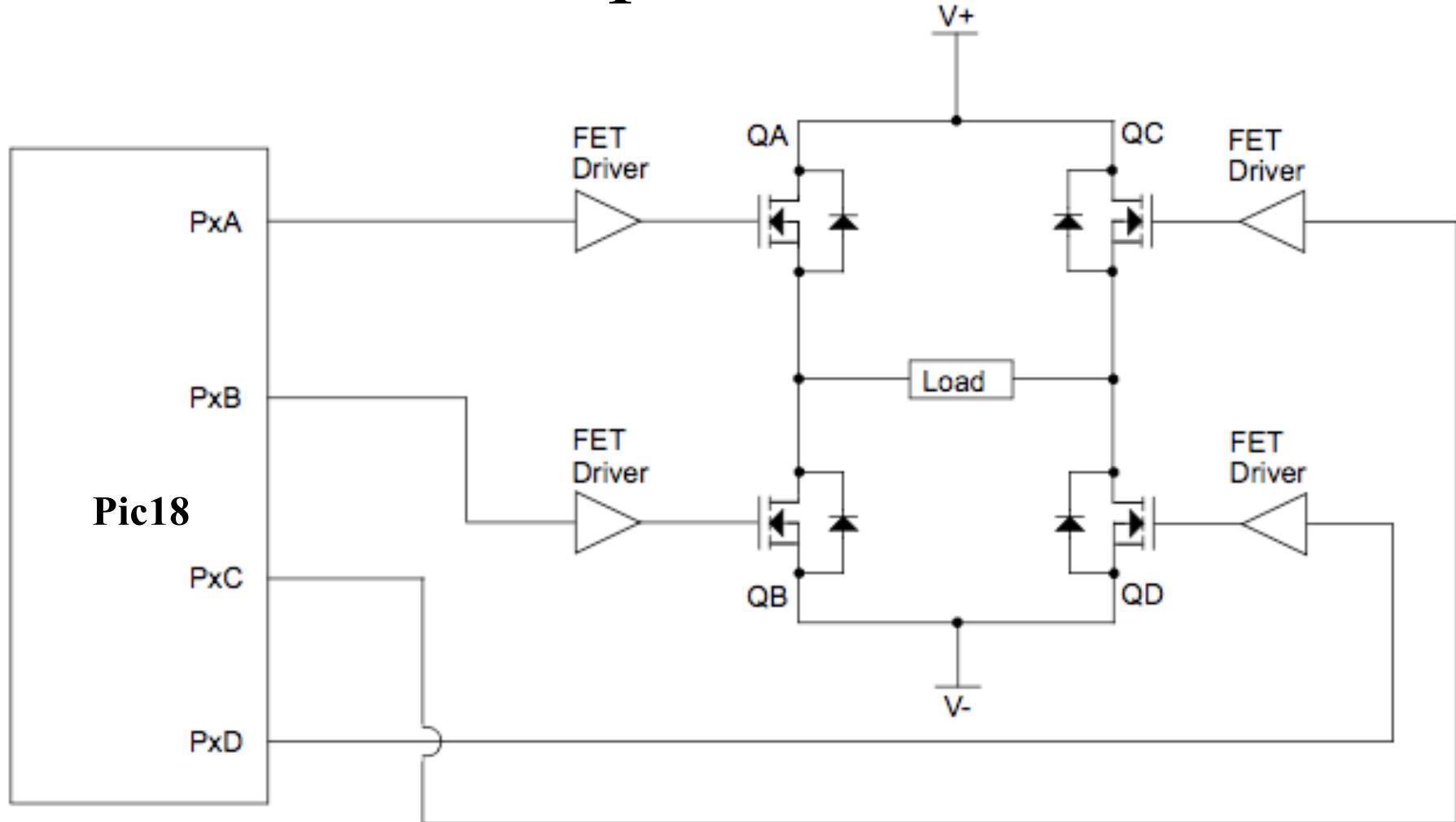


Free-wheeling Diodes

Zero Centered PW Modulation



PIC Microprocessor Control



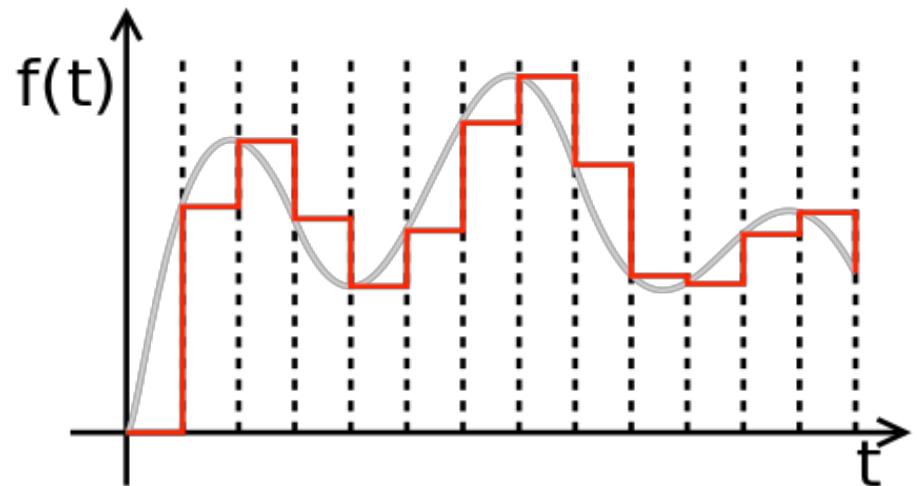
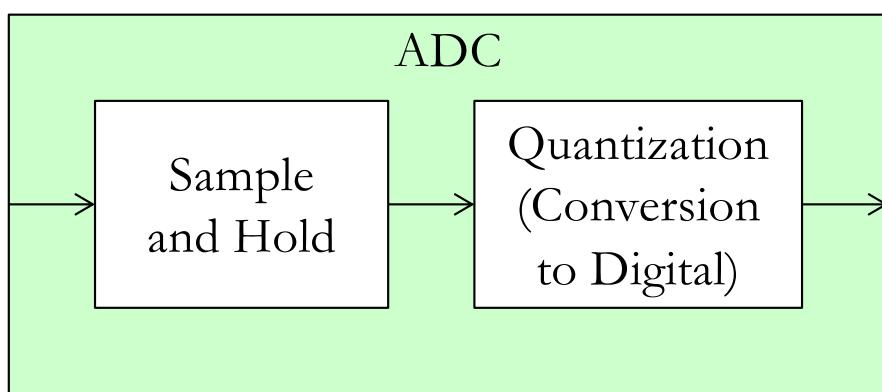
8-Bit microcontroller with built-in H-Bridge PWM controllers.

Lecture 26

Digital to Analog Conversion
Analog to Digital Conversion

Analog to Digital Convertor

- Abbreviations: ADC, A/D, A-to-D, A2D
- [One time] Samples a continuous physical quantity (usually voltage) and converts (quantizes) the sample to a digital number.
- [Continuous] Periodically samples a continuous physical quantity and converts (quantizes) the samples into a sequence of digital values.

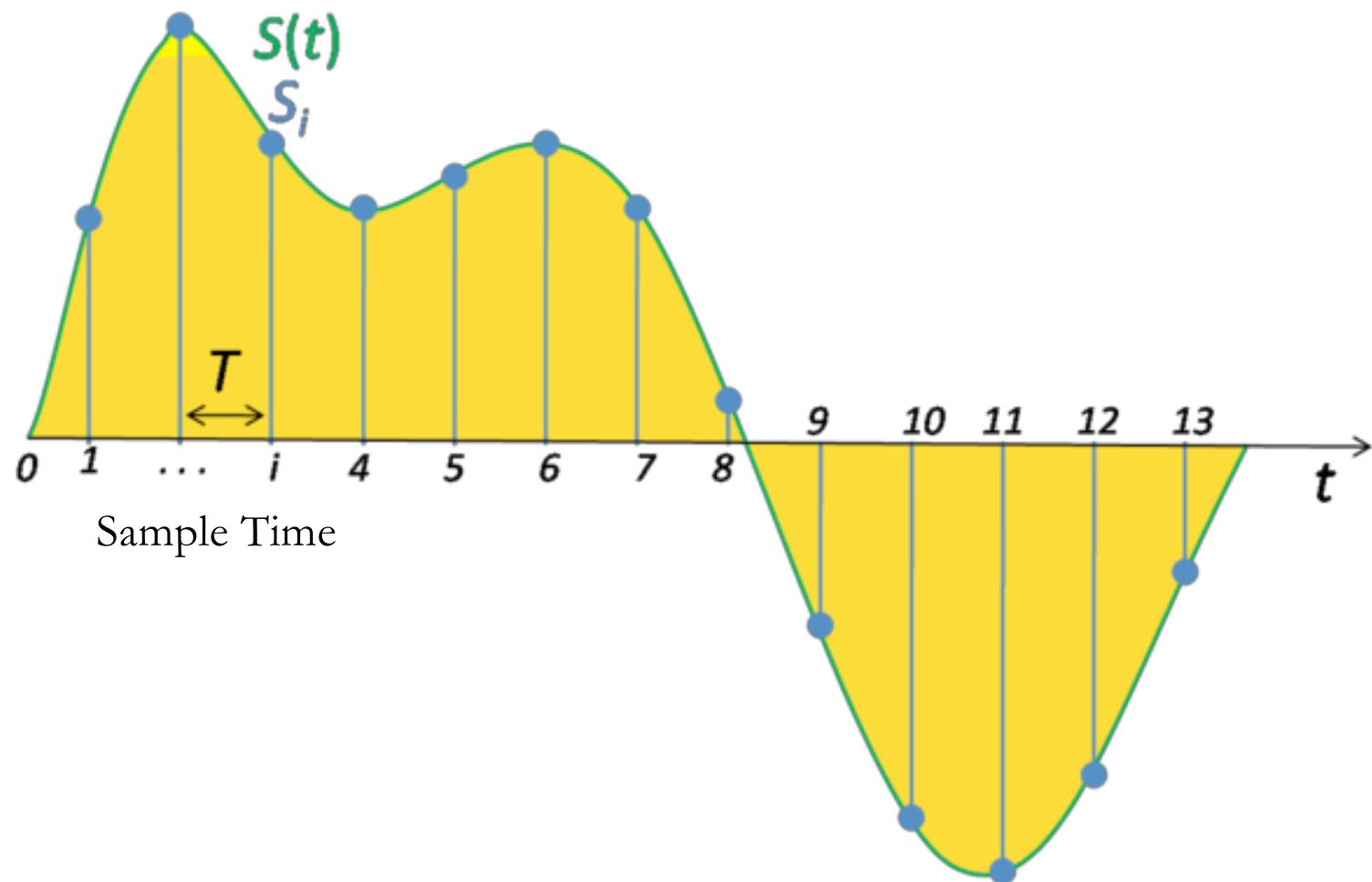


Characteristics of an ADC

- Sample rate – how often the waveform gets sampled.
- Resolution – n bits – 2^n is the number of output levels to which a signal can quantized.
 - Unsigned Range: 0 to $2^n - 1$, Signed range: -2^{n-1} to $2^{n-1} - 1$
Quantization error – rounding error introduced by quantization
- Linearity – how well the quantization levels match the true analog signal
- Accuracy – how much total error– given in $\pm k$ LSB
- Dynamic range – resolution of ADC reduced by noise from inaccuracies given in terms of effective number of bits

Sample Rate

- Sample rate (samples/sec) = $1/T$



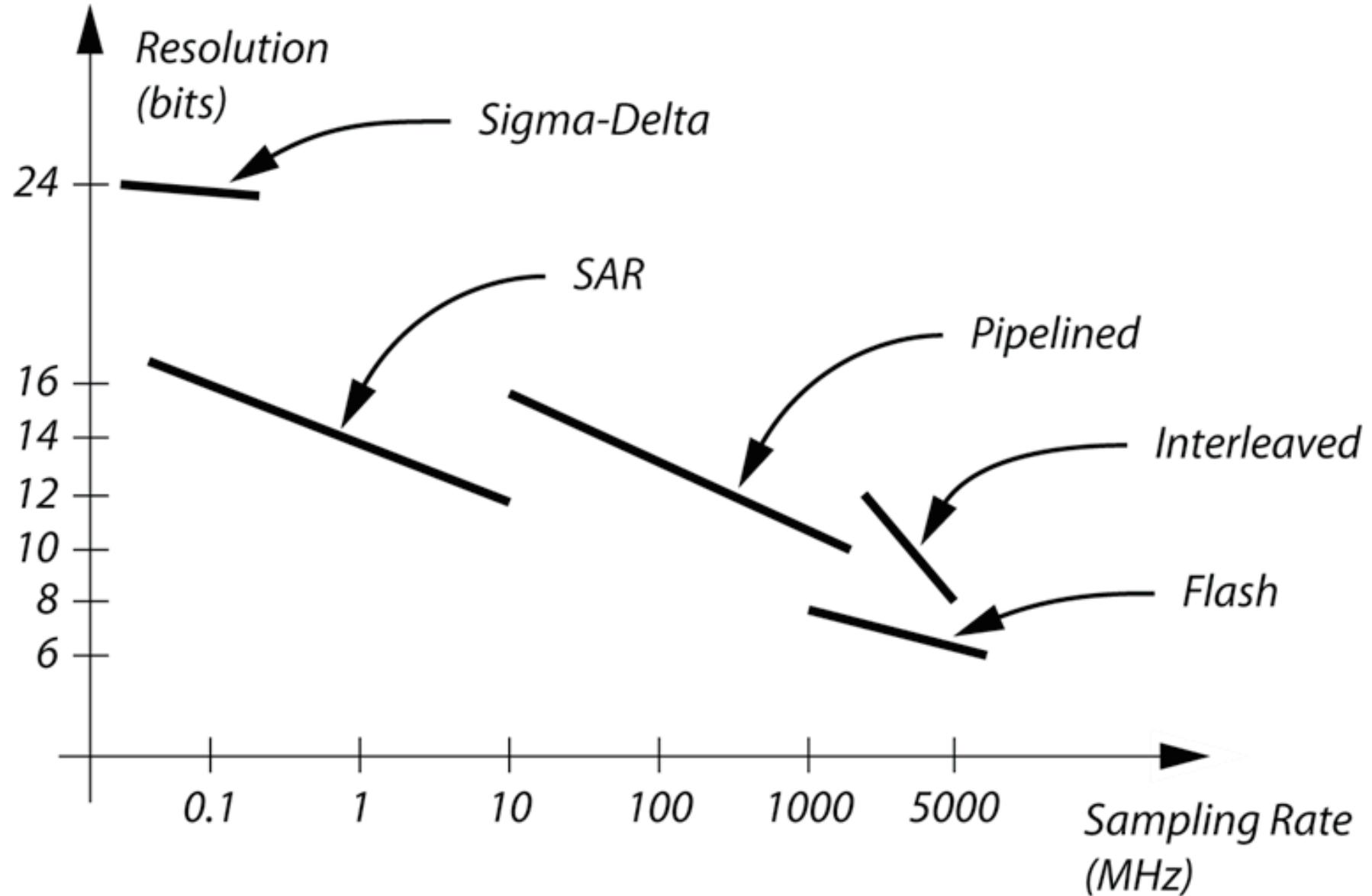
Resolution

Resolution (in bits)	Range	\pm Range	Quantization Error	Maximum Sampling Rate
24	16,777,216	\pm 8,388,608	\pm 0.000003%	200 KHz
16	65,536	\pm 32,768	\pm 0.0008%	250 MHz
14	16,384	\pm 8,192	\pm 0.003%	400 MHz
12	4,096	\pm 2,048	\pm 0.012%	1.8 GHz
10	1,024	\pm 512	\pm 0.05%	2.2 GHz
8	256	\pm 128	\pm 0.2%	3 GHz
6	64	\pm 32	\pm 0.8%	6 GHz

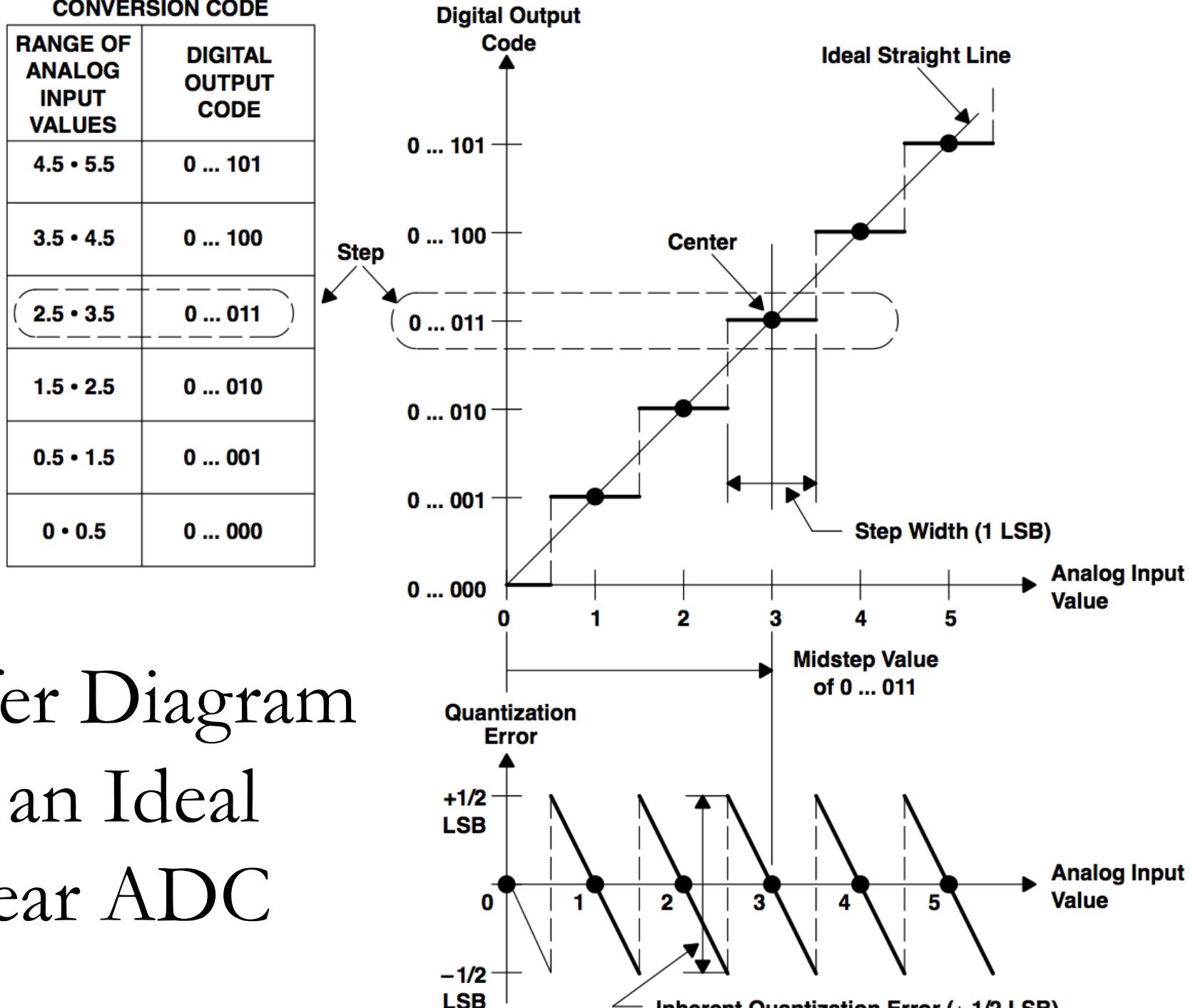
This shows the tradeoff between resolution and maximum sampling rate

The most common ADC techniques currently on the market and where they sit in
the spectrum of

Resolution versus Sampling Rate.

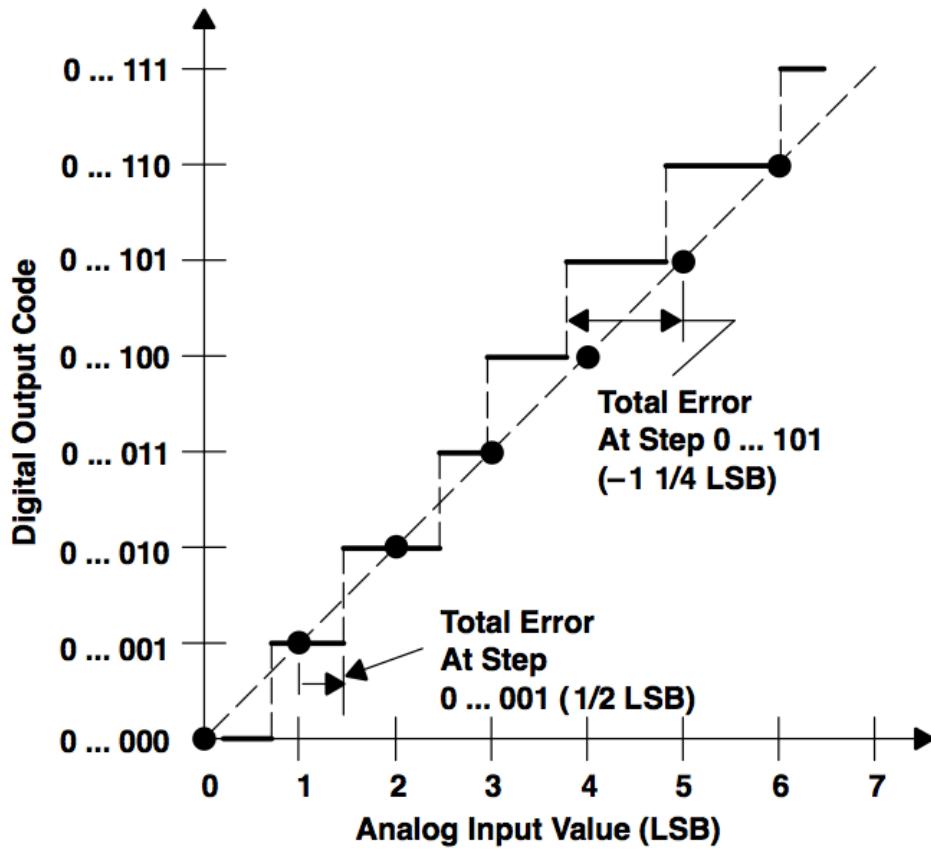


Transfer Diagram for an Ideal Linear ADC

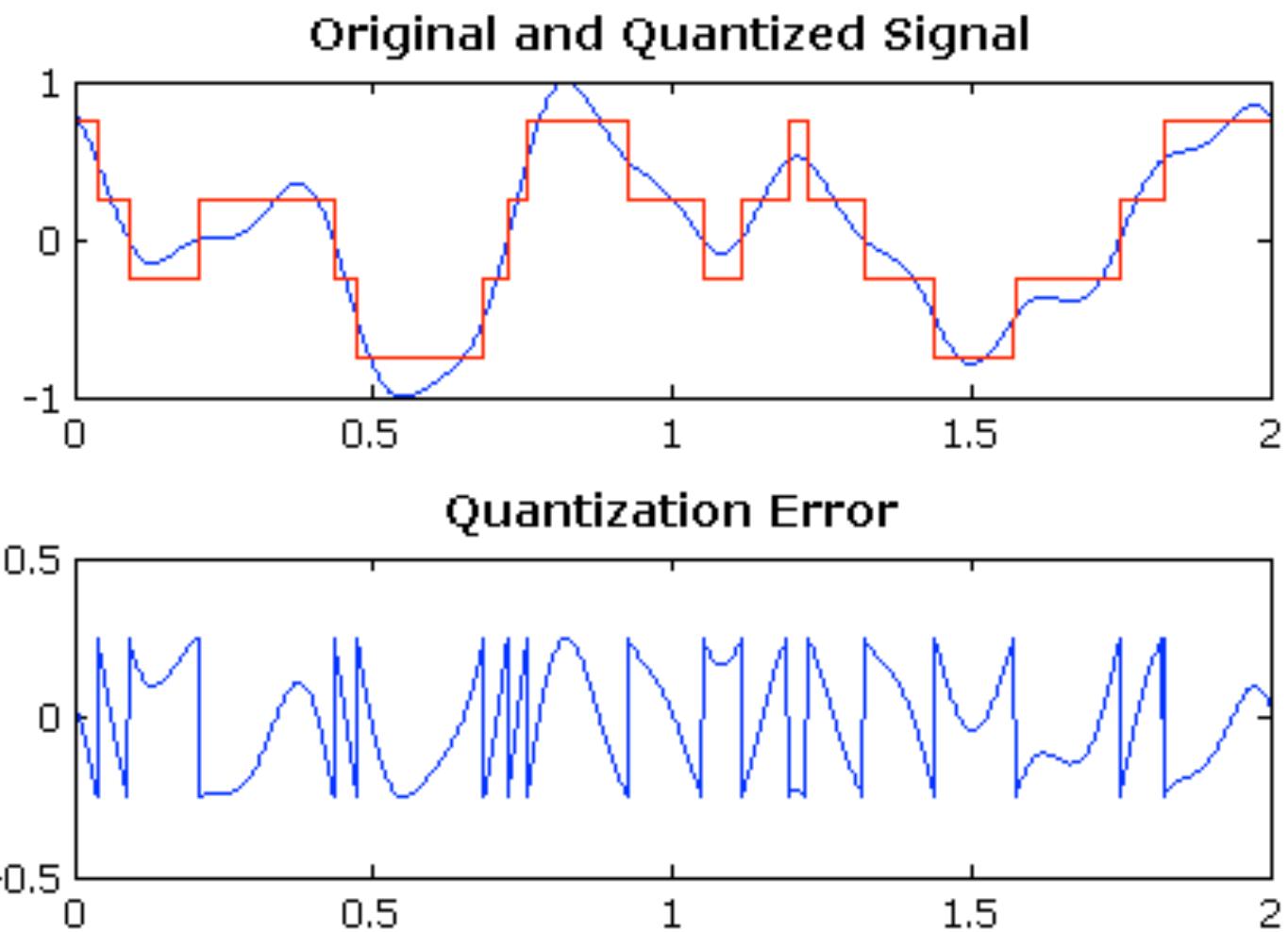


Elements of Transfer Diagram for an Ideal Linear ADC

Non-Linearity Errors

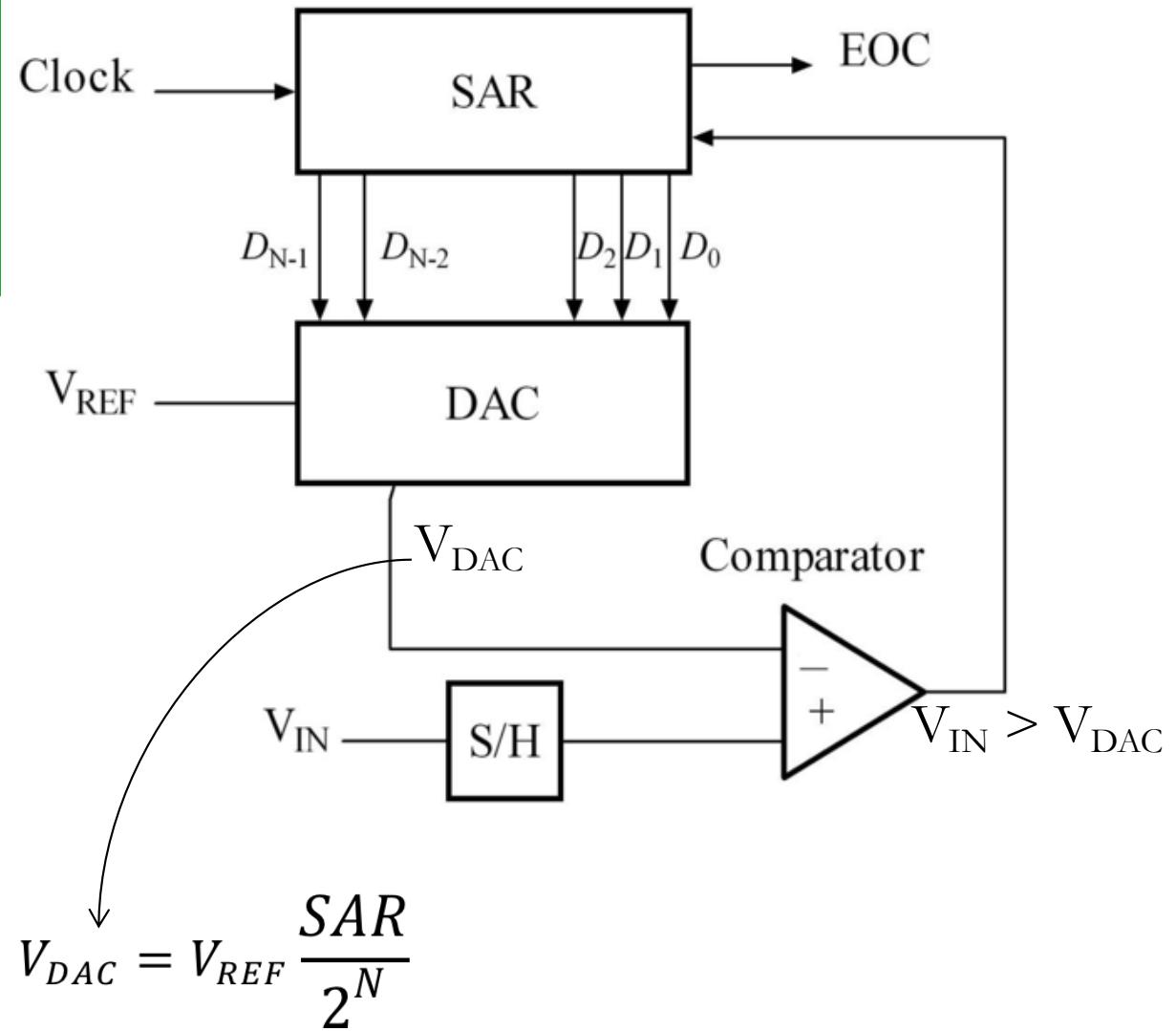
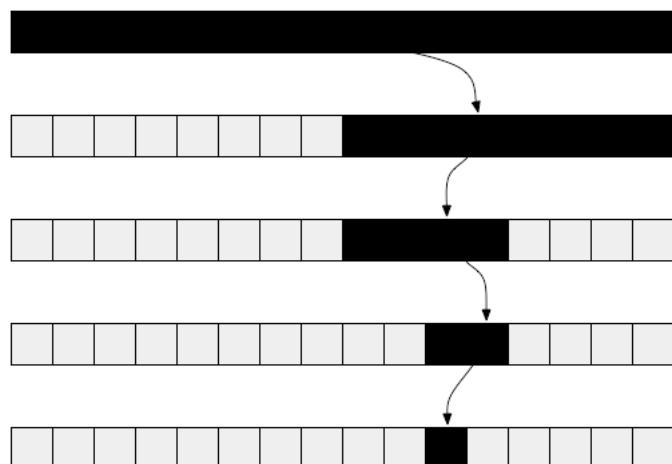


Quantization Error



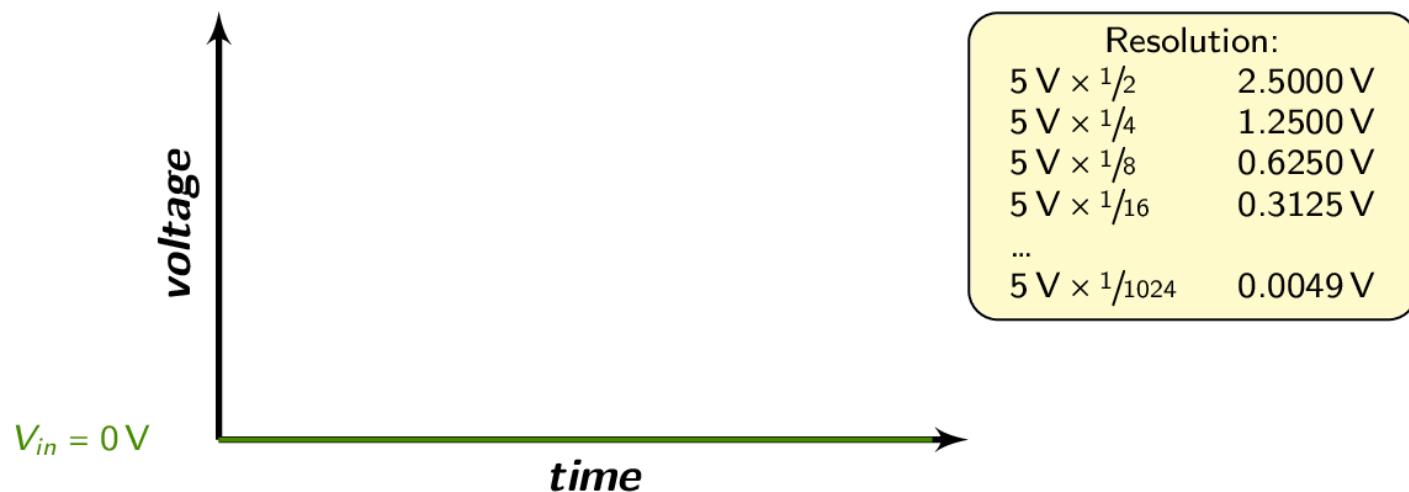
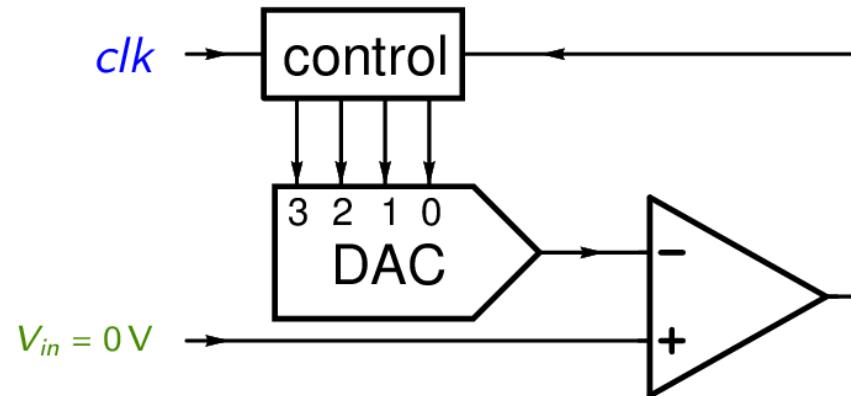
Successive Approximation Register (SAR) Type ADC

Finds the digital value by a binary search— one bit at a time starting with the Most Significant Bit (MSB).



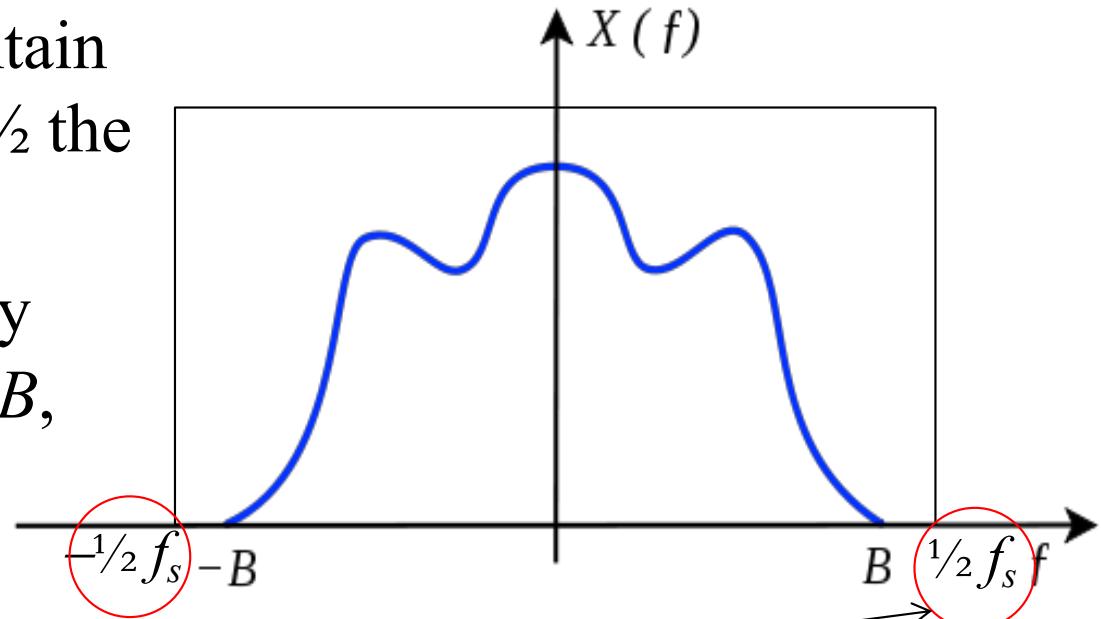
Successive Approximation Register (SAR) Type ADC

Successive Approximation – example of a 4-bit ADC

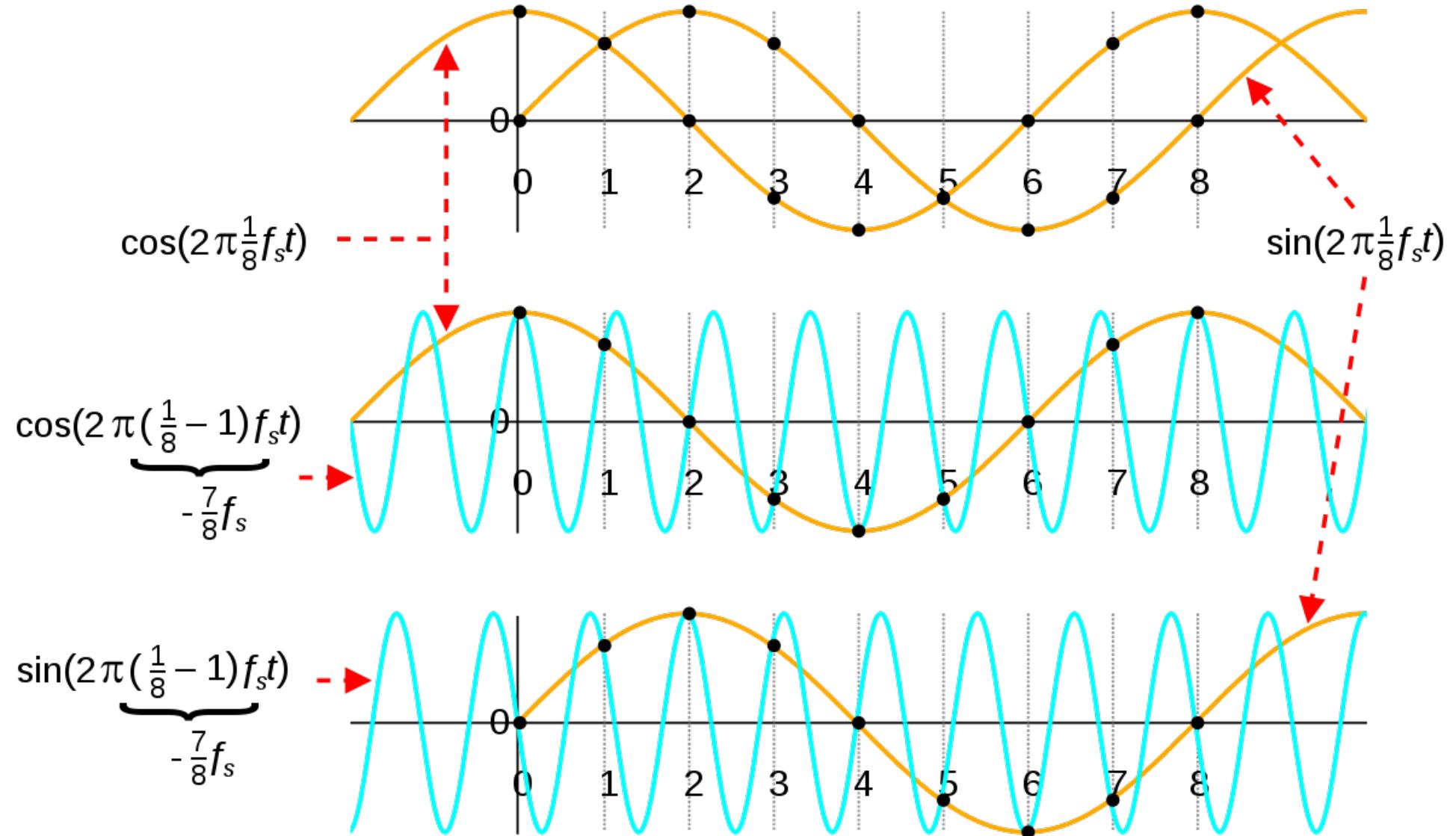


Nyquist–Shannon Sampling Theorem

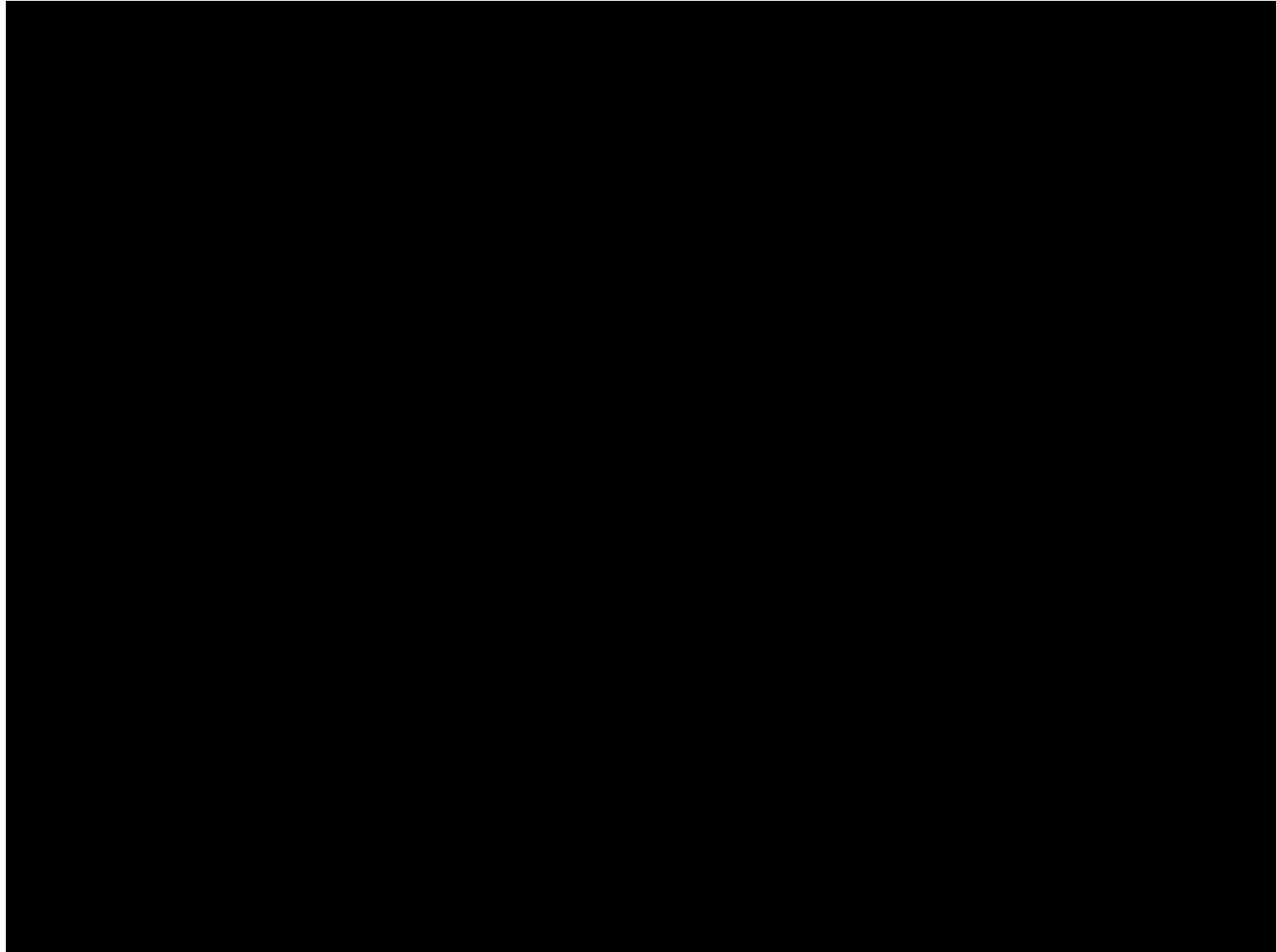
- This theorem is the fundamental bridge between continuous signals (*analog* domain) and discrete signals (*digital* domain).
- It answers this question: “How fast must you sample a signal
 - in order to not lose any information?
 - in order to be able to fully reconstruct the signal from its samples?
- A: The signal $x(t)$ must contain no sinusoidal frequency $\geq \frac{1}{2}$ the sample rate f_s .
- A: If signal $x(t)$ is frequency band-limited by frequency B , then $B < \frac{1}{2}f_s$.
- The “Nyquist frequency” = $\frac{1}{2}$ the sample rate.



Nyquist–Shannon Sampling Theorem



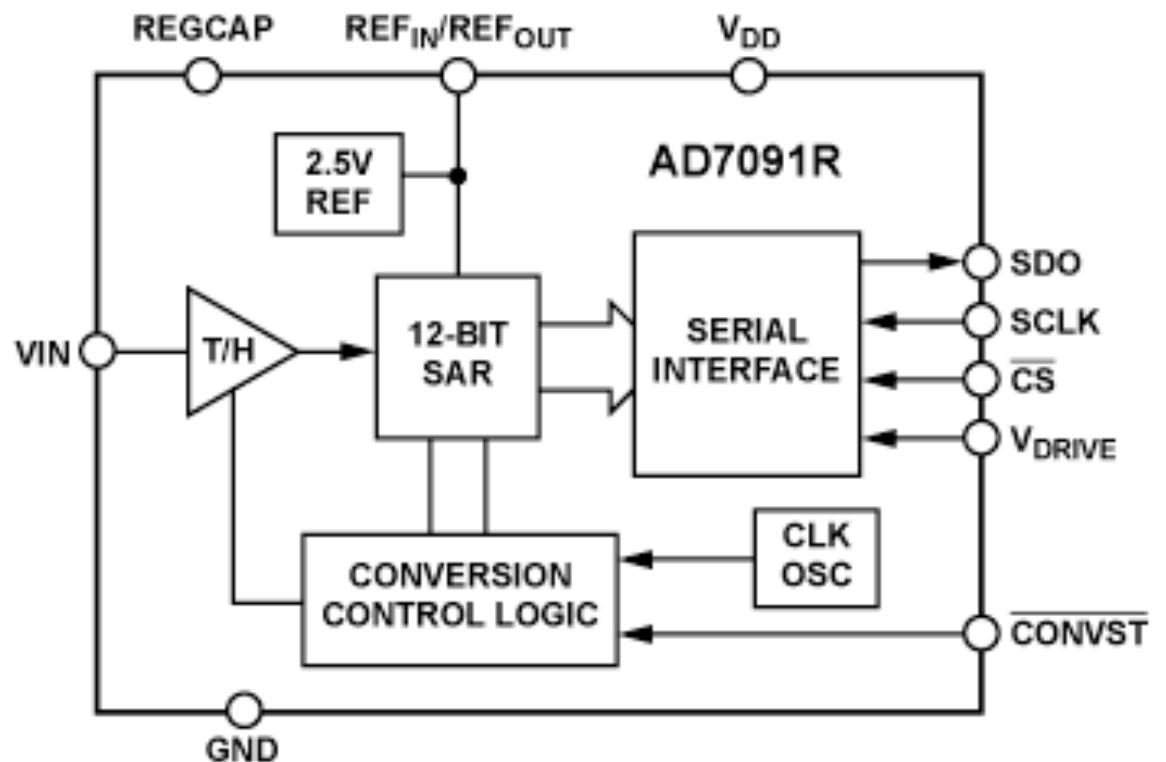
Nyquist–Shannon Sampling Theorem



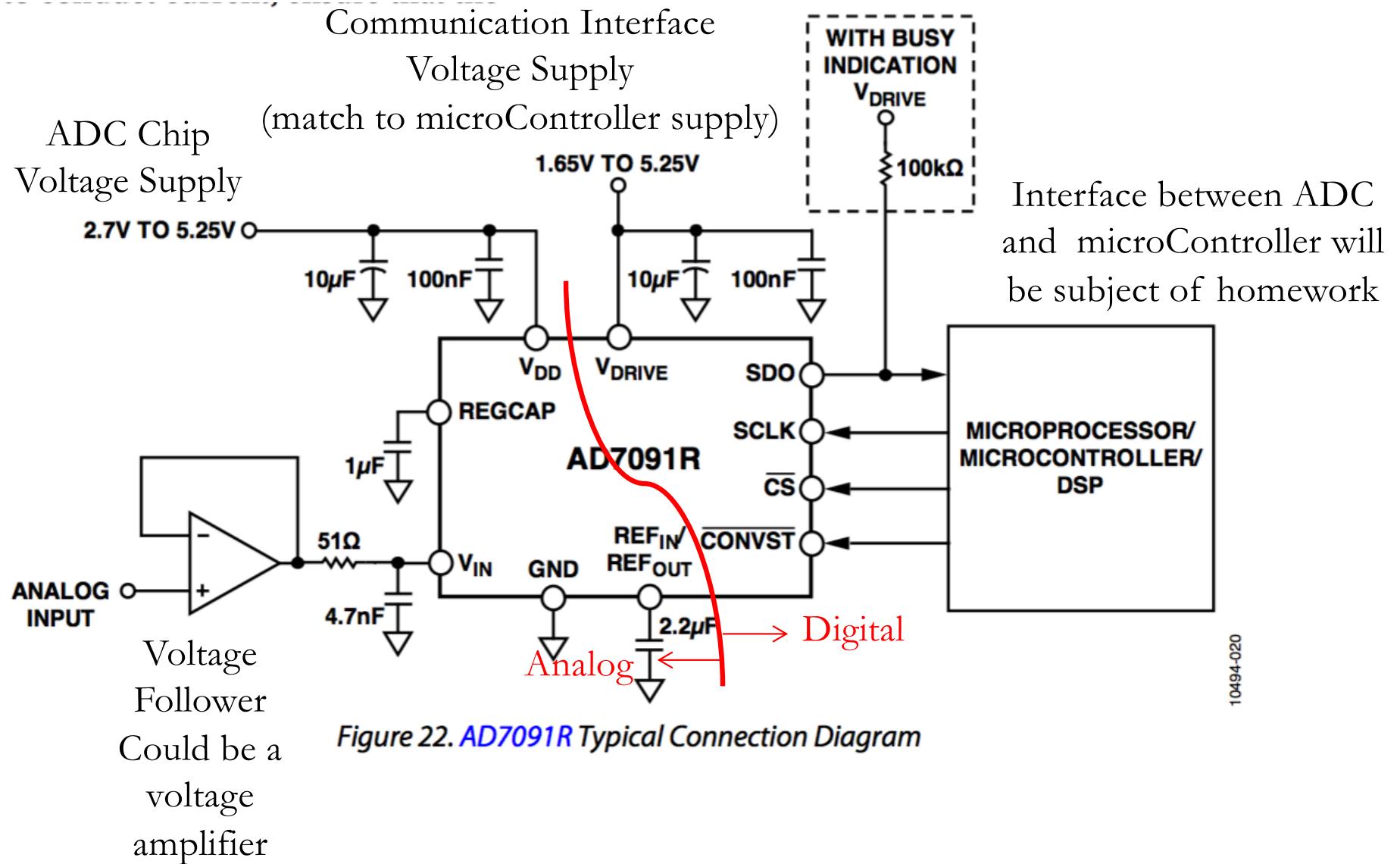
Using an ADC

Analog Devices AD7091R

- 12 bit (resolution)
- 1 MSPS (million samples per second)
- Internal 2.5 volt reference voltage
- Ultra low power

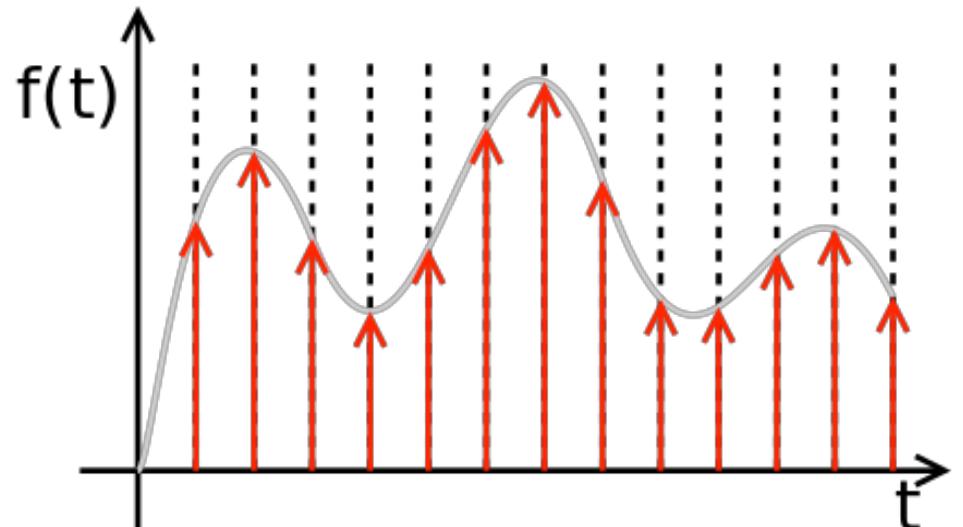
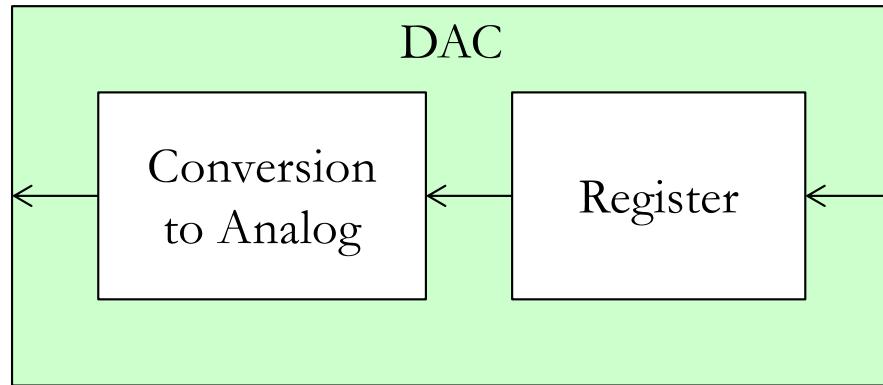


Circuitry Around the ADC



Digital to Analog Converter

- Abbreviations: DAC, D/A, D-to-A, D2A
- [One time] Converts a digital number to an analog signal, usually current or voltage.
- [Continuous] Converts a sequence of digital numbers to an analog waveform.

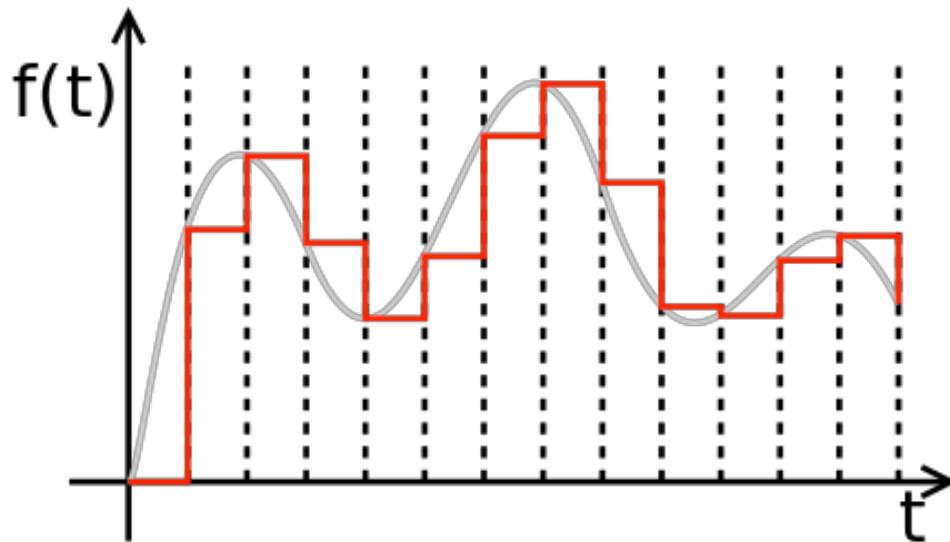


DAC Characteristics

- DACs are characterized in a similar way to ADCs.
- Speed – Samples per second
- Resolution – n Bits
- Accuracy – $\pm k$ LSB

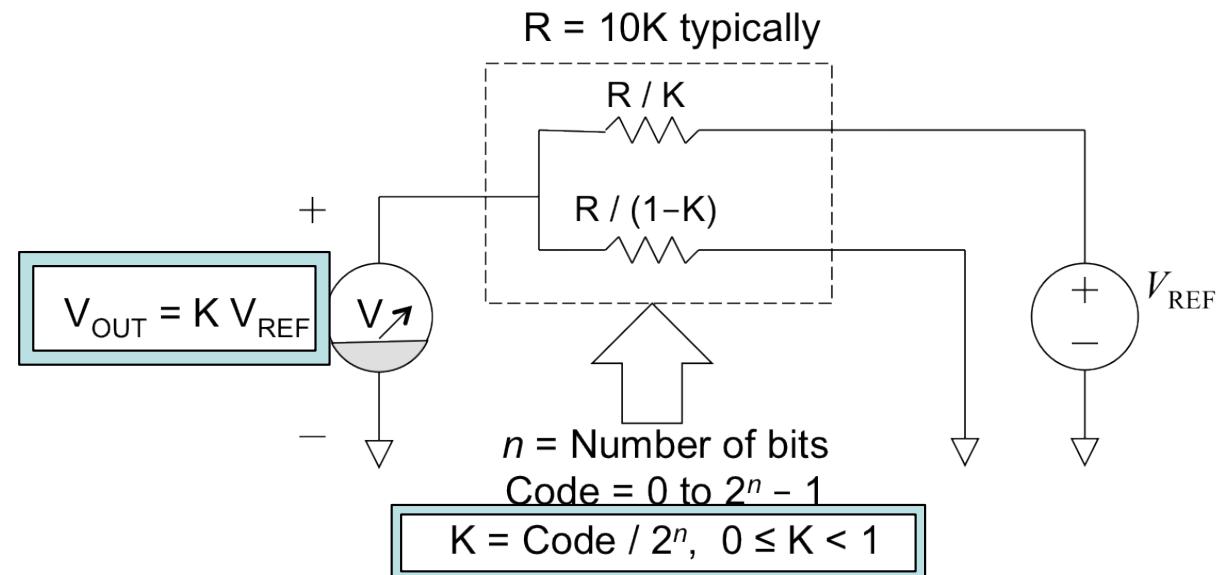
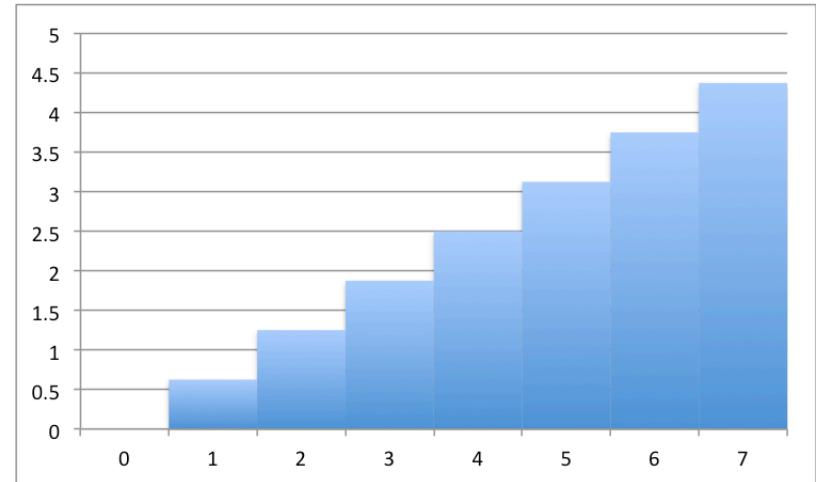
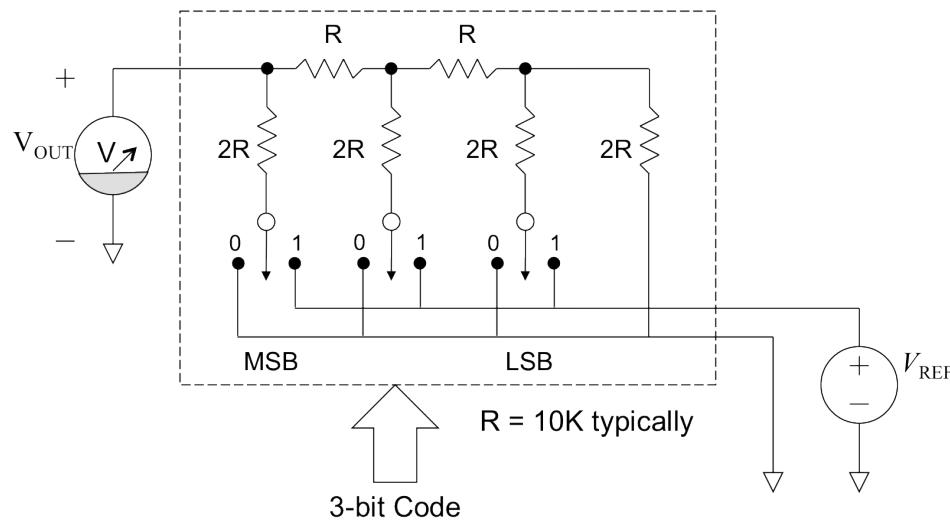
Signal Reconstruction

- The output of the DAC is (the red waveform).

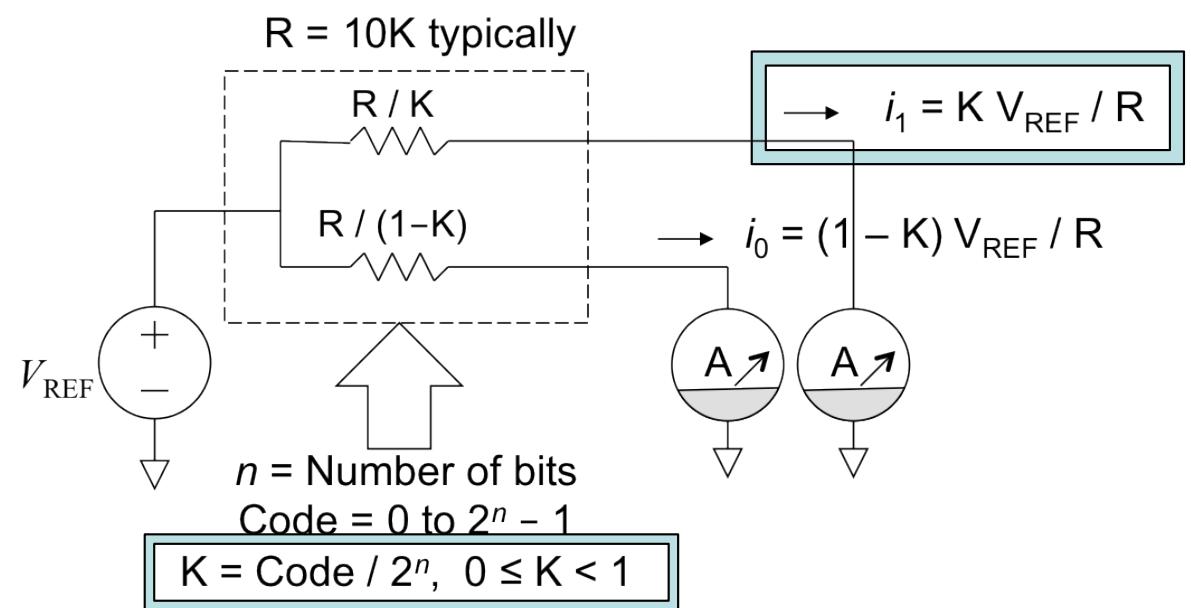
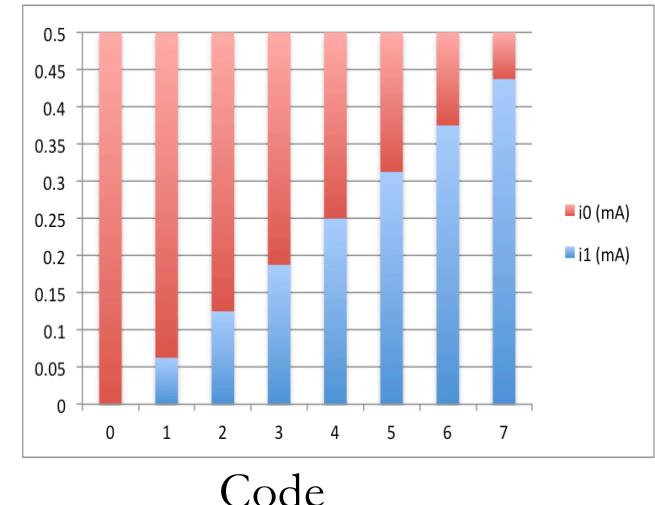
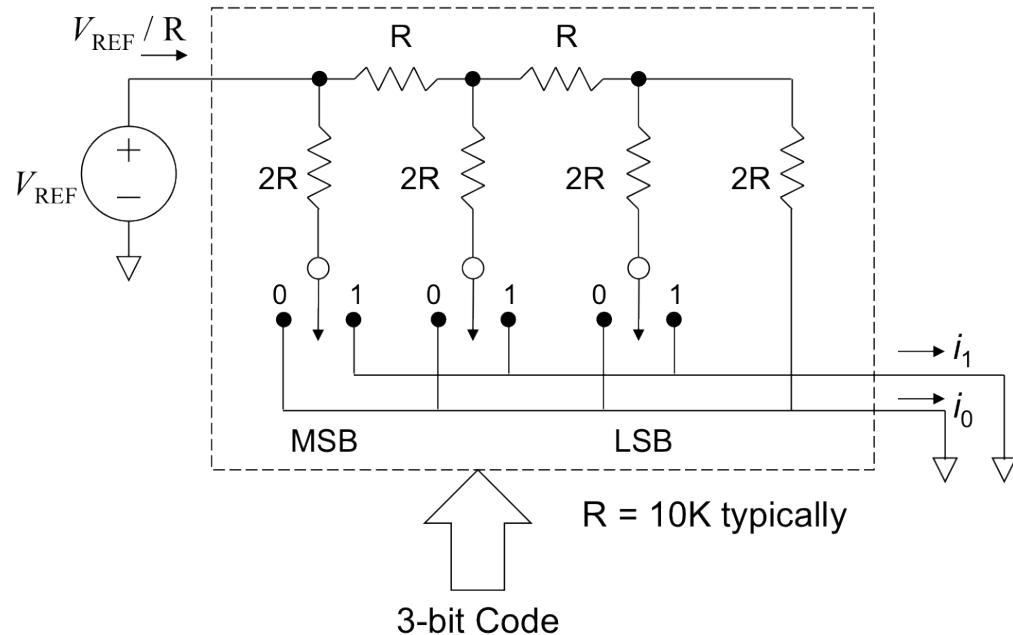


- To reconstruct the signal (the grey waveform), you have to filter the output of the DAC with a Low Pass Filter (reconstruction filter) with cutoff frequency < Nyquist frequency.

R-2R Ladder: Voltage Mode (review)



R-2R Ladder: Current Mode (review)

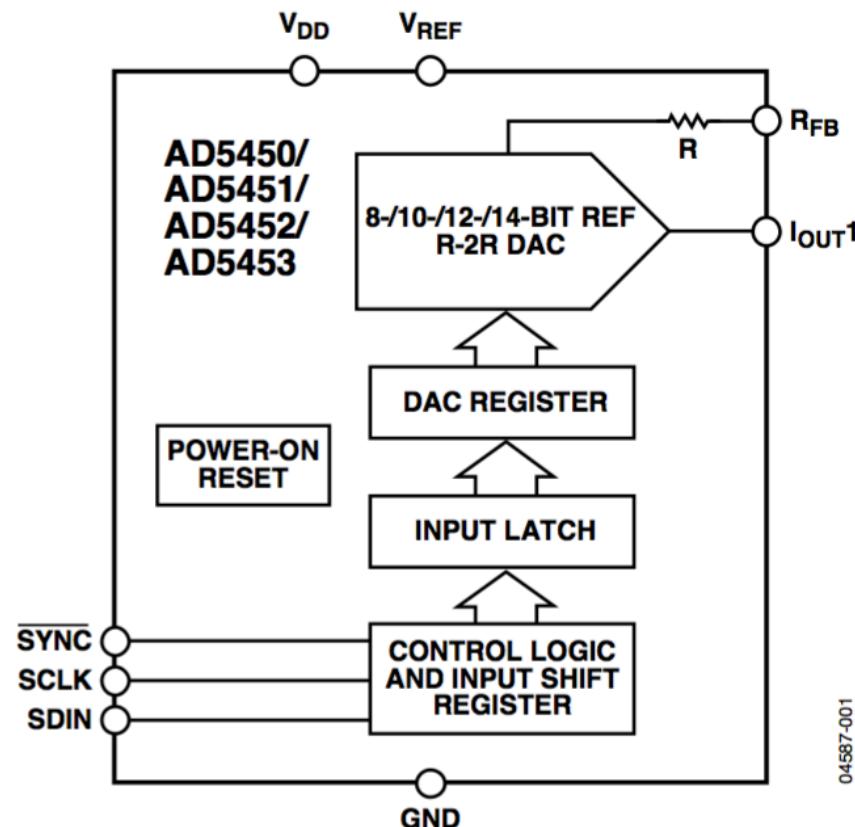


Using a DAC

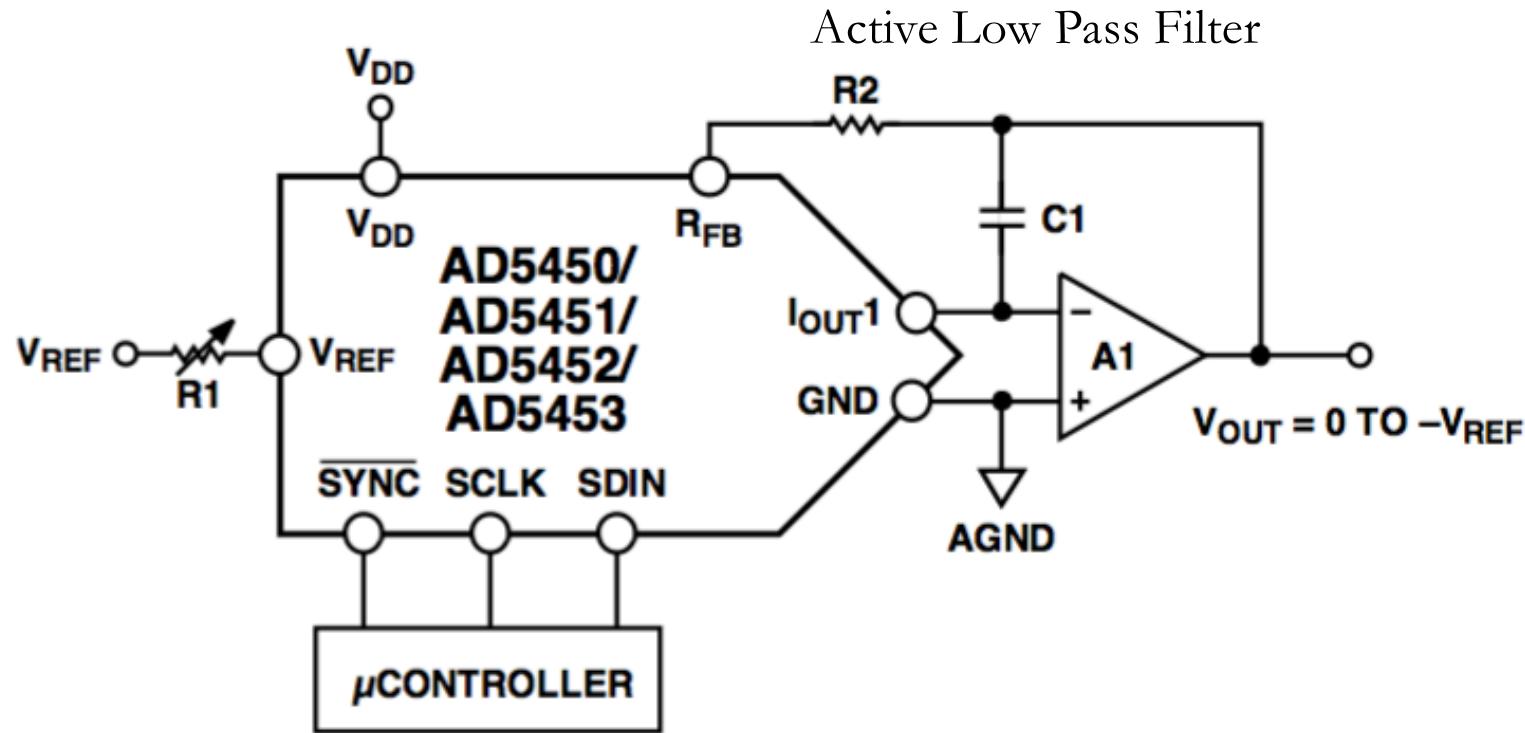
Analog Devices AD5452

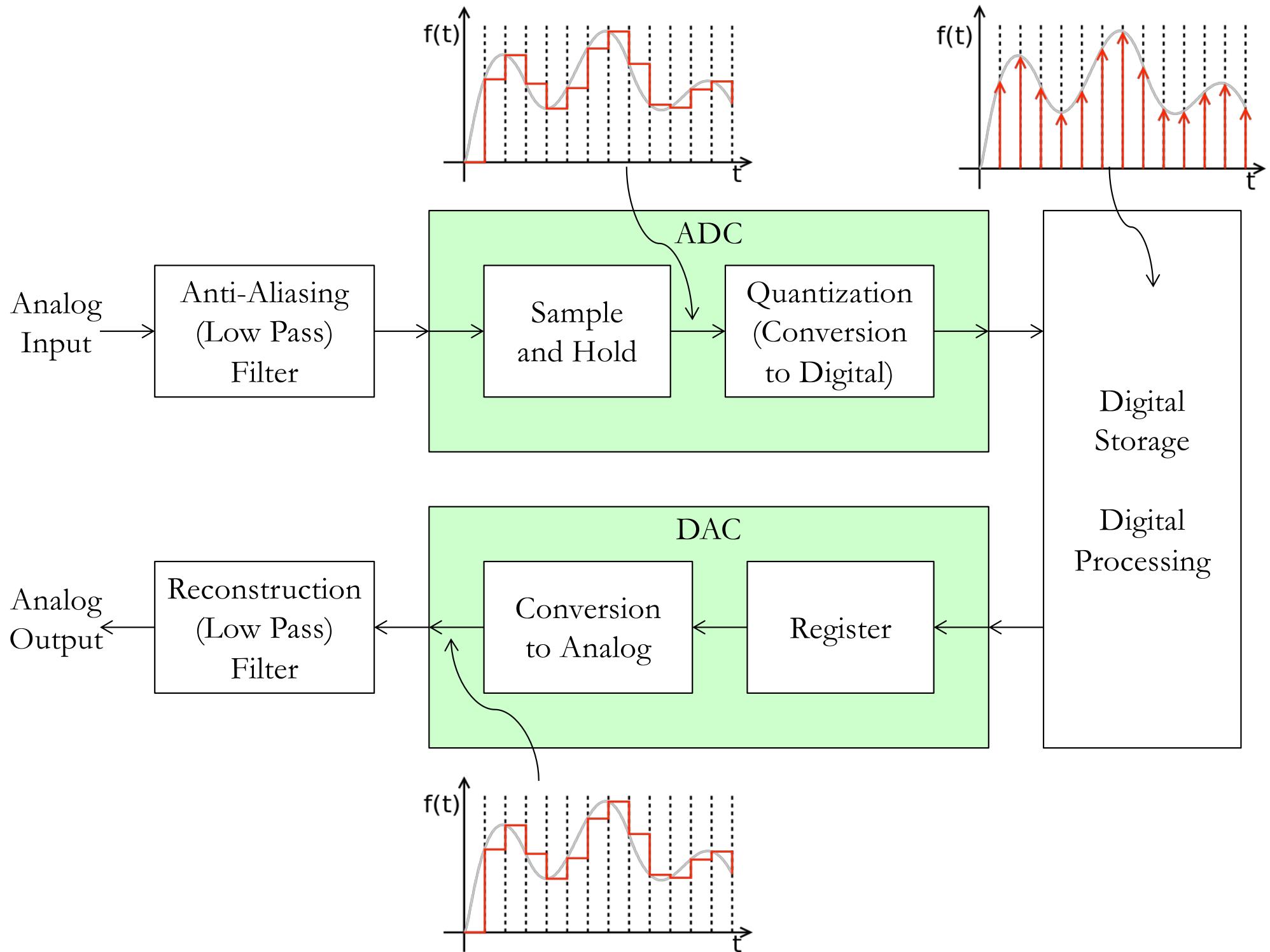
- 12 bit (resolution)
- 2.7 MSPS (million samples per second)
- Requires external reference voltage
- Uses R-2R Ladder

FUNCTIONAL BLOCK DIAGRAM



Circuitry Around the DAC





ADC and DAC Summary

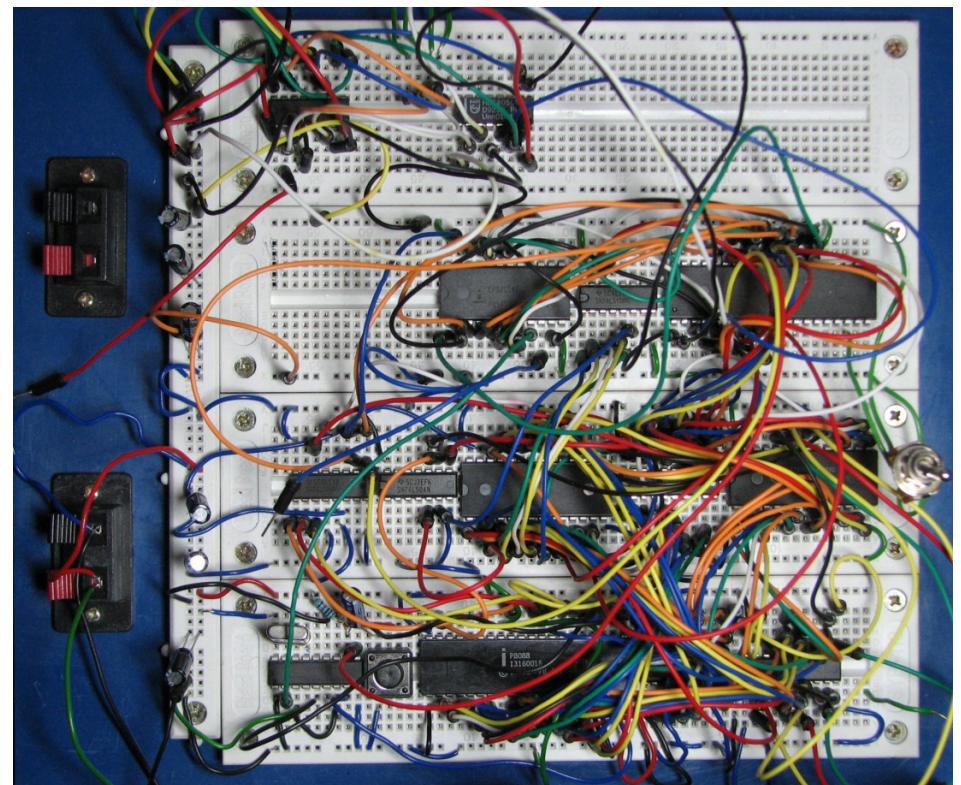
- ADCs and DACs are the interface between:
 - continuous analog waveforms, and
 - discrete digital signals.
- ADCs convert sensor information to digital values.
- DACs convert digital values to analog voltages to control motors, actuators, etc.
- Allow computers (microControllers) to be in the control loops of non-digital systems such as mechanical or chemical systems.

Lecture 27

Printed Circuit Board Design

Printed Circuit Boards (PCBs)

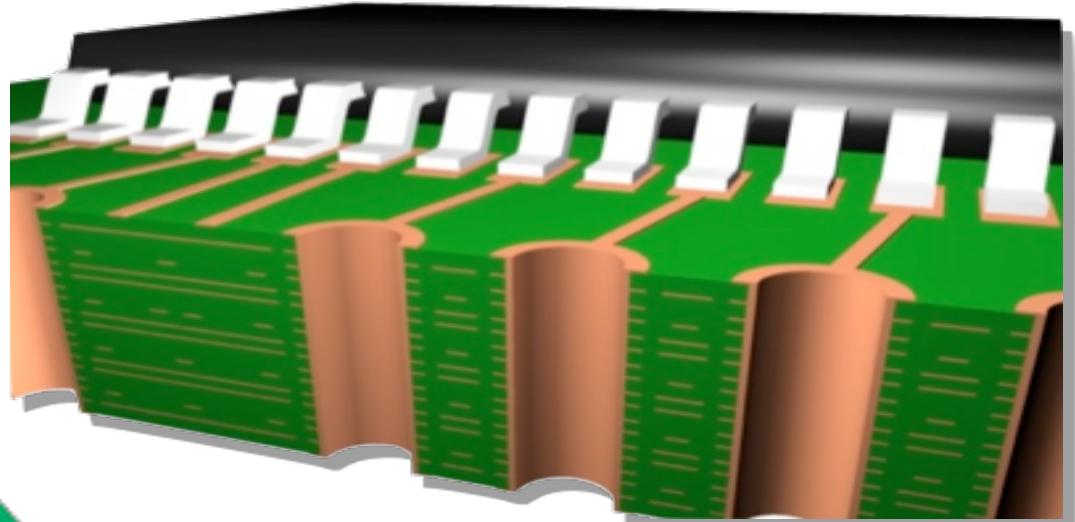
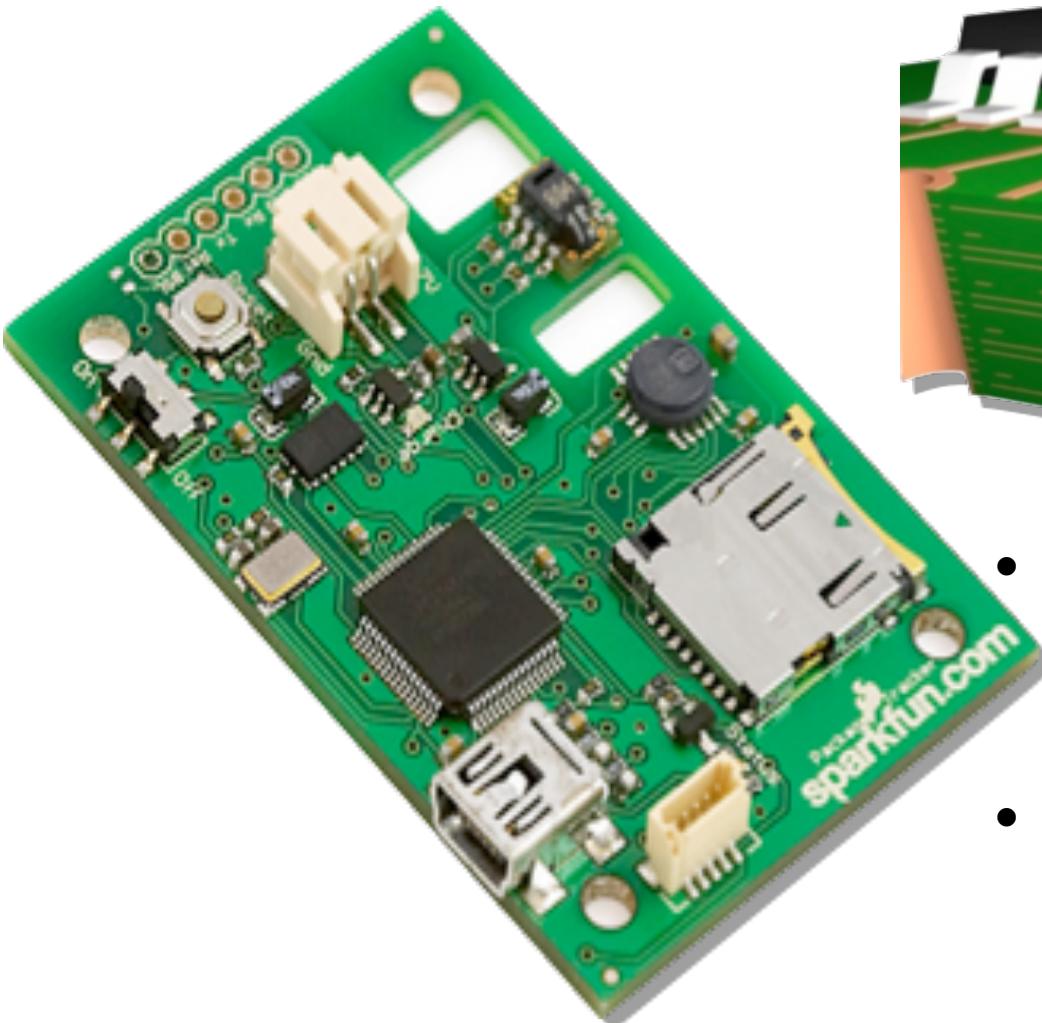
- Features
 - Surprisingly Affordable
 - Highly reliable
 - Compact
- Drawbacks
 - Requires more layout than other board types
 - Higher initial cost than wire wrap or point-to-point construction



Less Desirable Alternative to a PCB

What is a PCB?

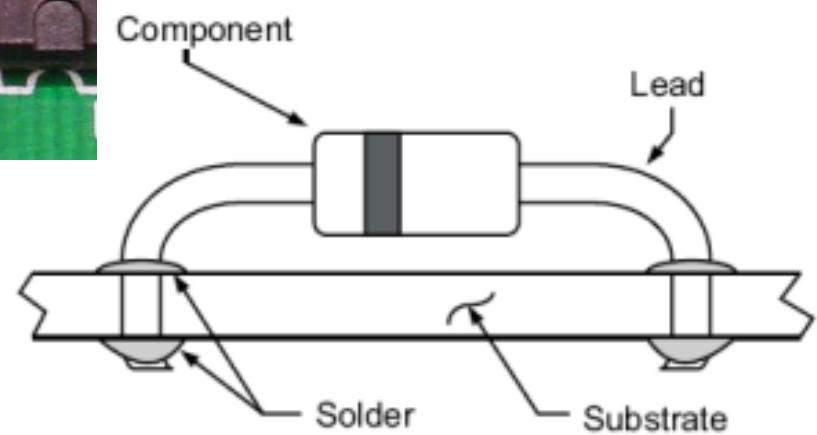
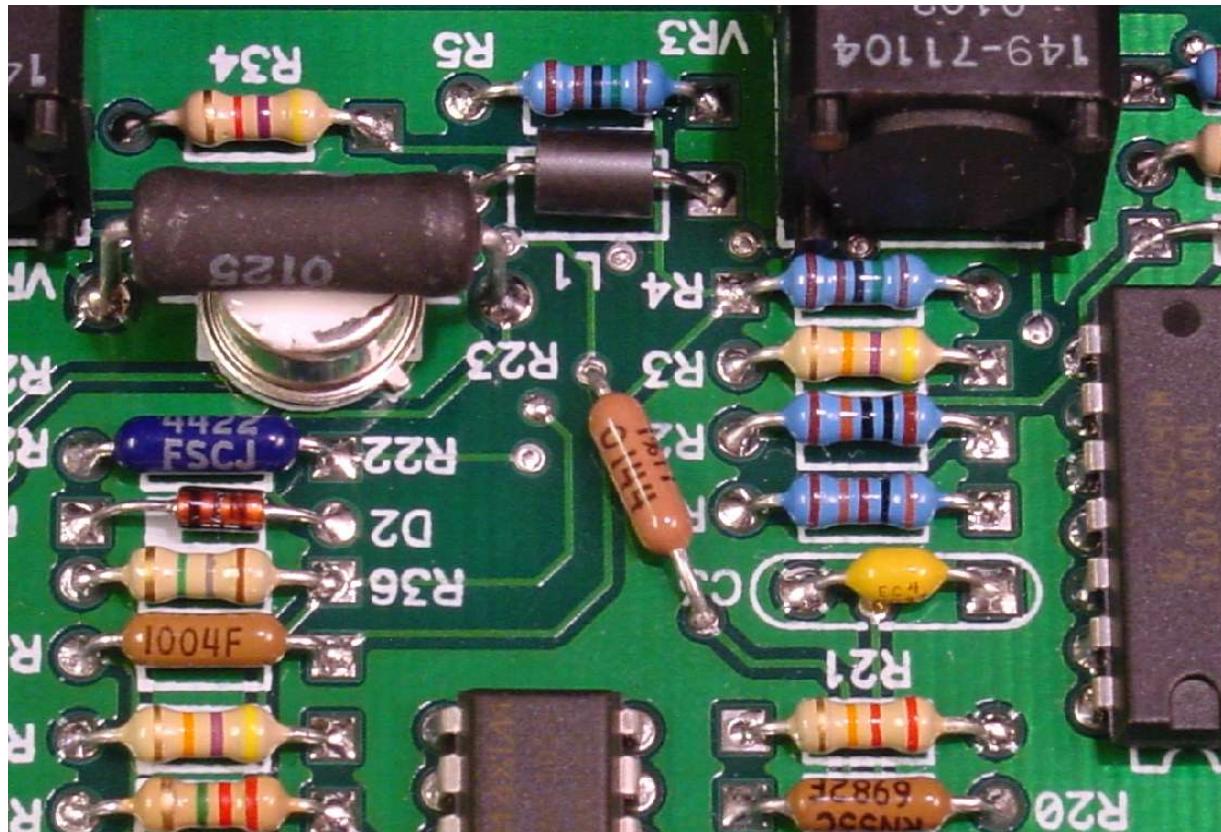
Layers, Traces, and Vias



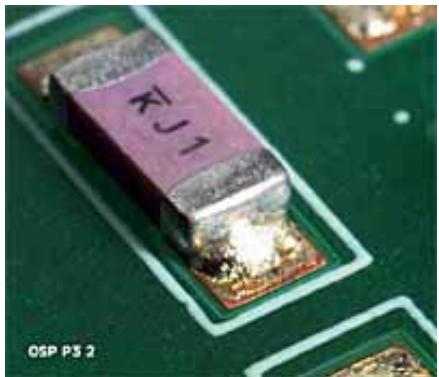
- Circuit components are interconnected using board traces and vias.
- The board is made of alternating layers of conducting and insulating materials

Image from www.elkosoft.com

Through-Hole Components



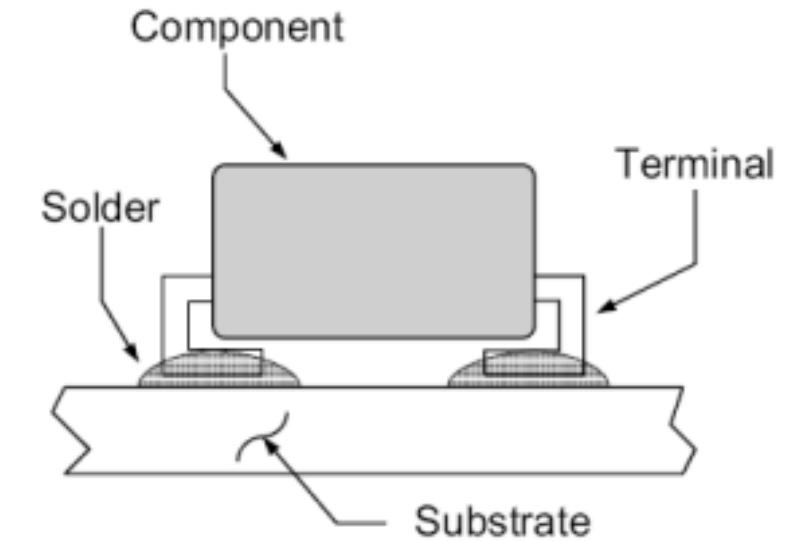
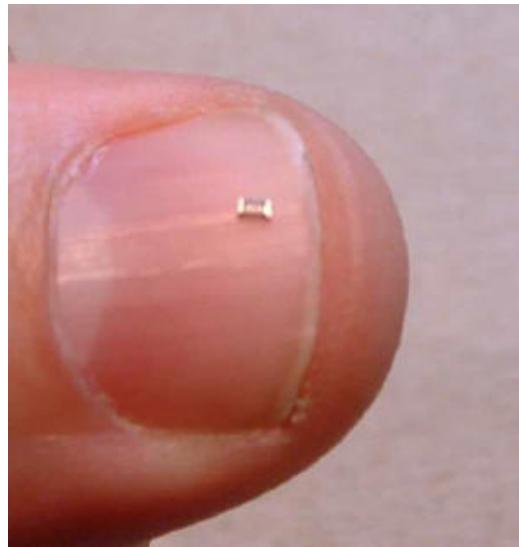
Surface-Mount Components



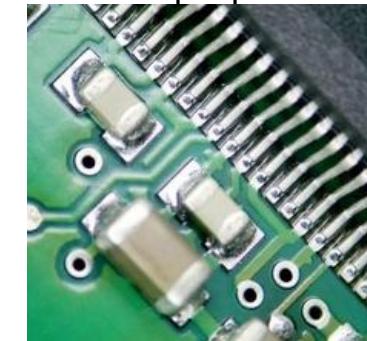
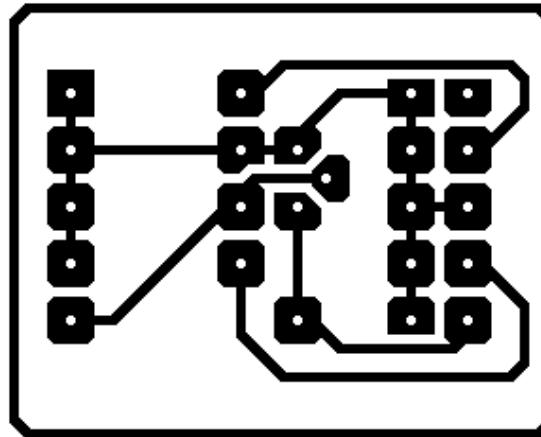
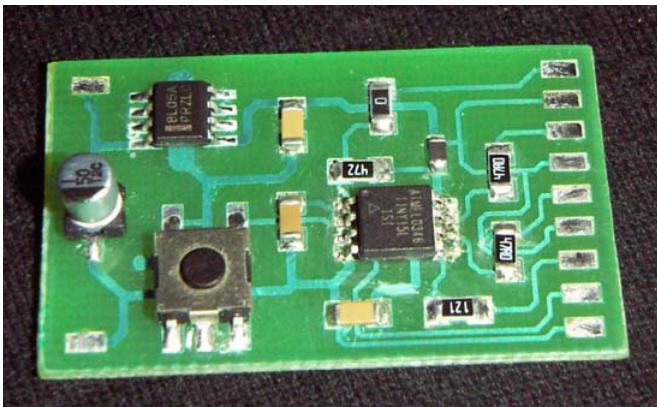
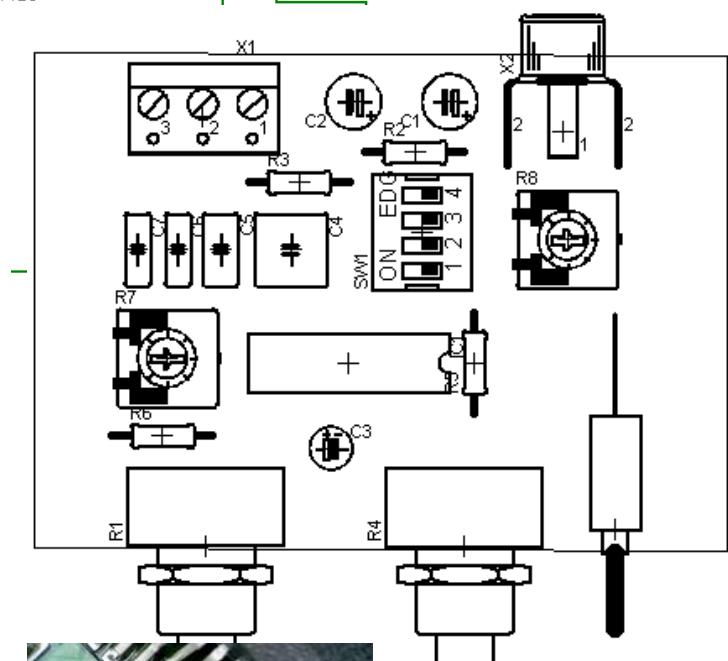
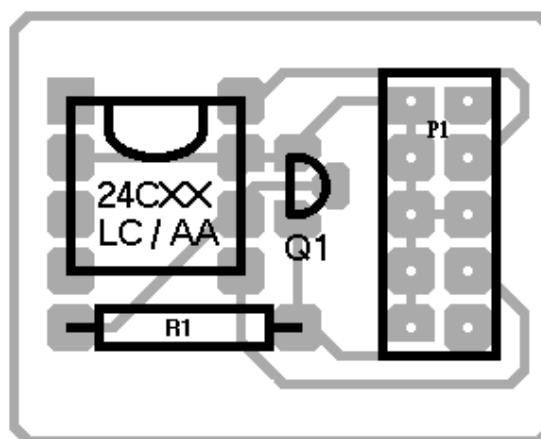
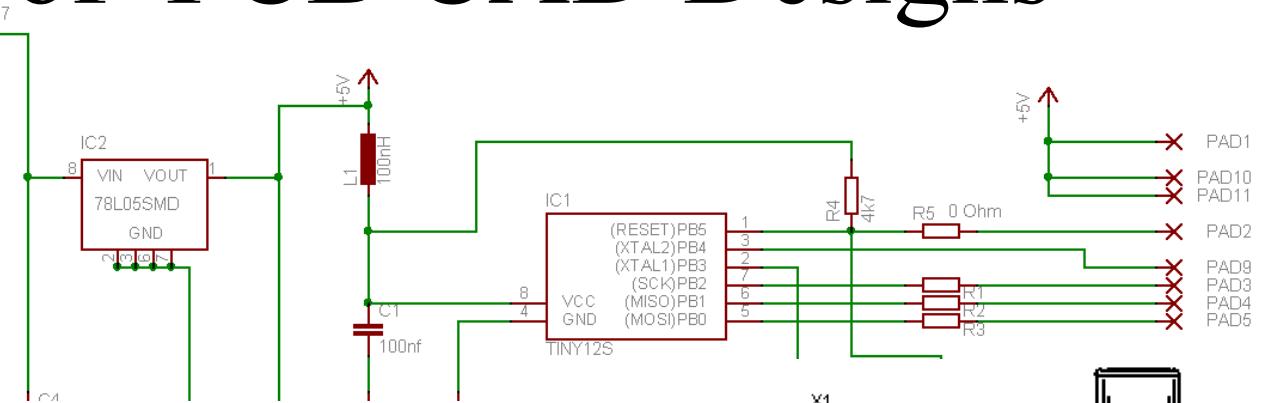
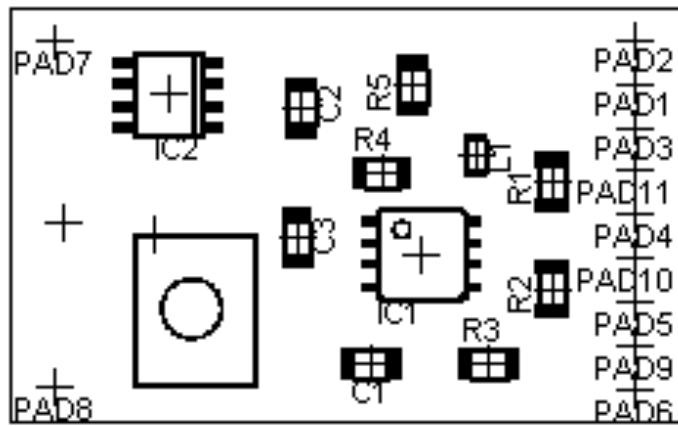
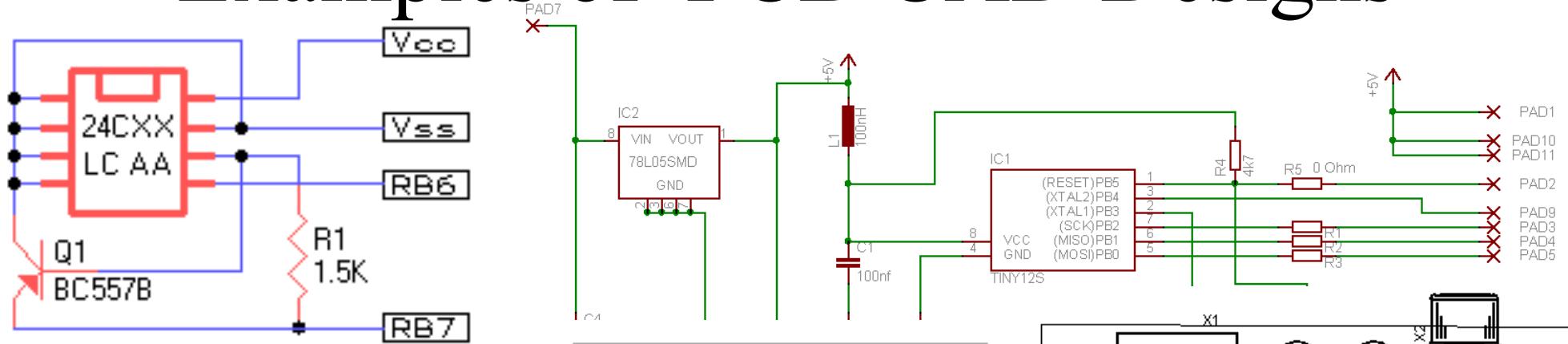
SMT technology is newer, much smaller, and cheaper to assemble.



Figure 6. iMEMS accelerometer in surface mount package



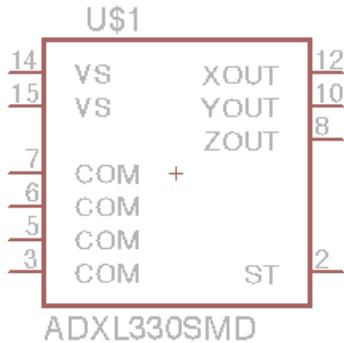
Examples of PCB CAD Designs



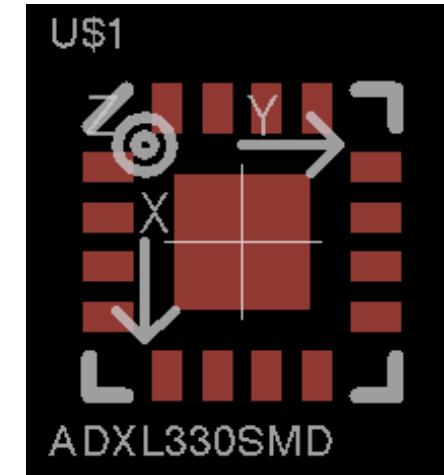
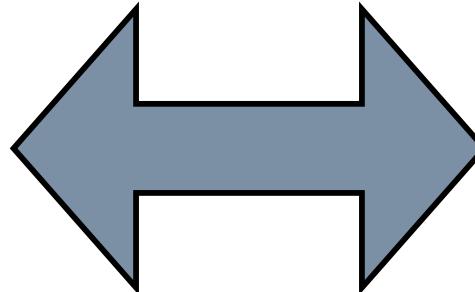
PCB Design Steps

- Design the Circuit
 - Create library components
 - Read data sheet to identify pins
 - Draw the symbol
 - Draw the package
 - Draw the schematic
 - Place the part symbols
 - Draw the nets and buses
- Design the PCB
 - Place the components
 - Route the signals
 - Check the board (DRC)
- Create PCB manufacturing (CAM) data files

Create Library Components

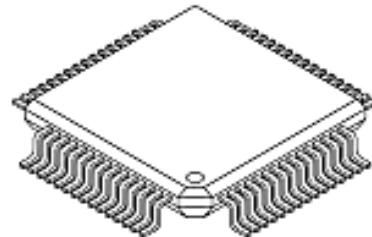


Schematic Symbol

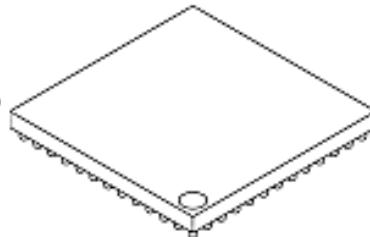


Component Footprint

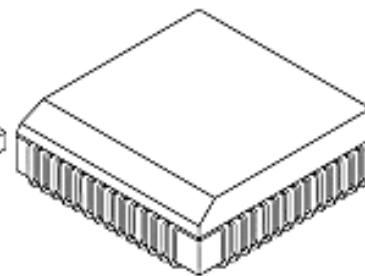
[QFP](#)



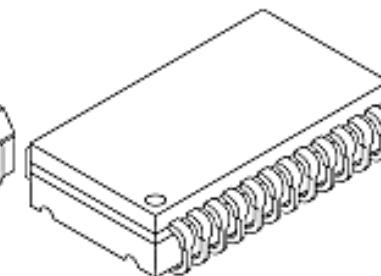
[BGA](#)



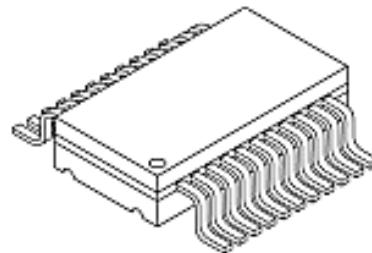
[PLCC](#)



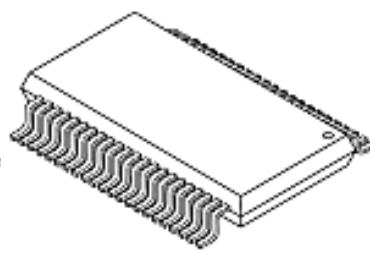
[SOJ](#)



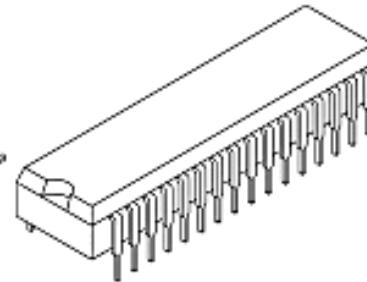
[SOIC](#)



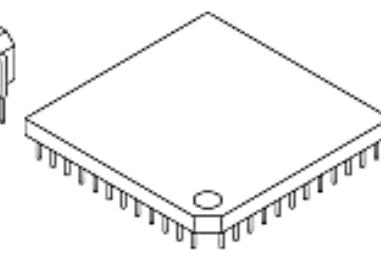
[TSOP](#)



[DIP](#)



[PGA](#)



Component Footprint Design

Dimensions and
Pin Numbers

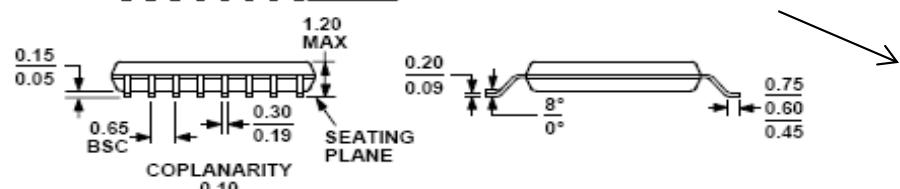
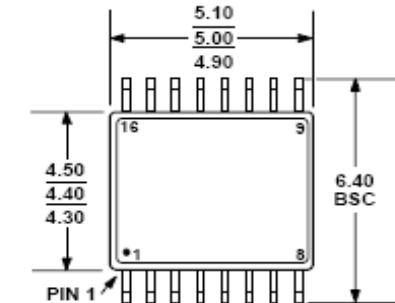
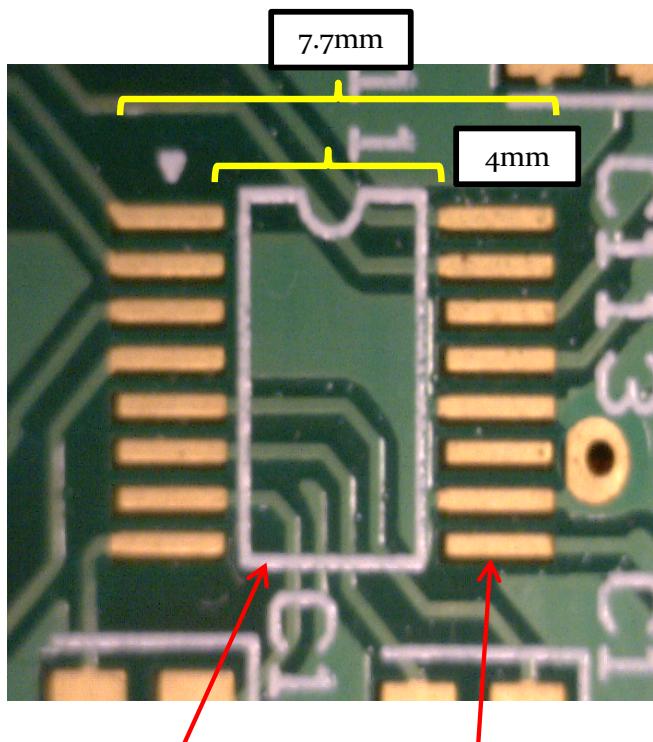


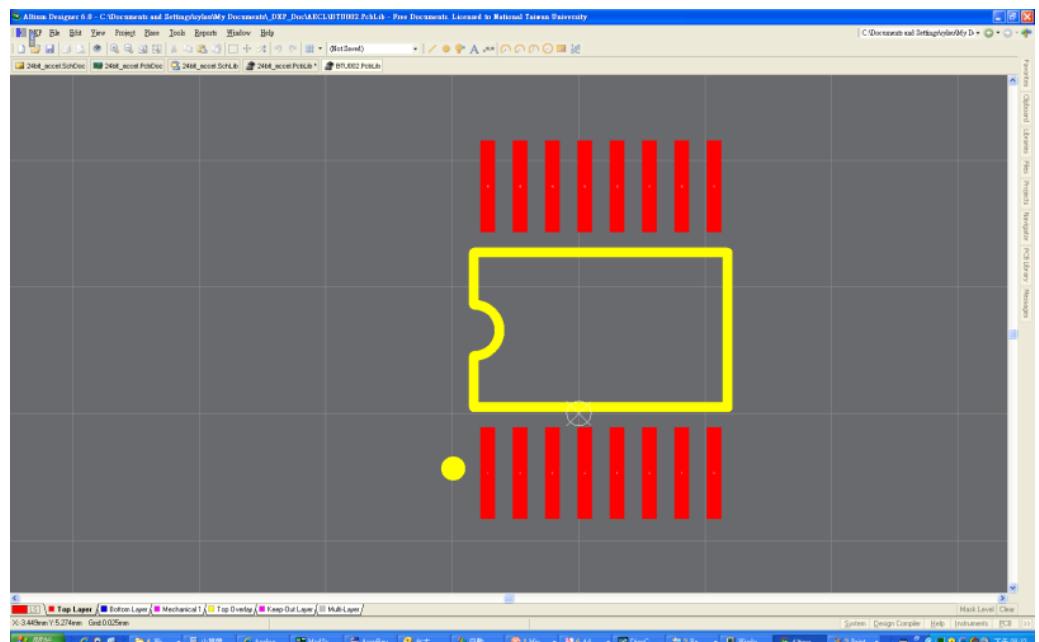
Figure 20. 16-Lead Thin Shrink Small Outline Package [TSSOP]
(RU-16)

Dimensions shown in millimeters



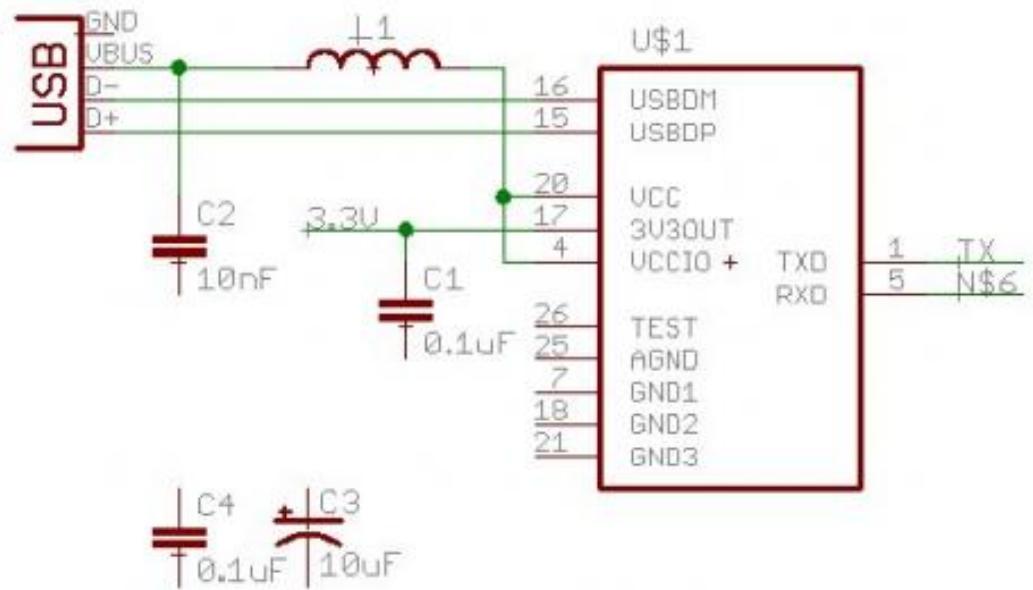
Overlay

Pads



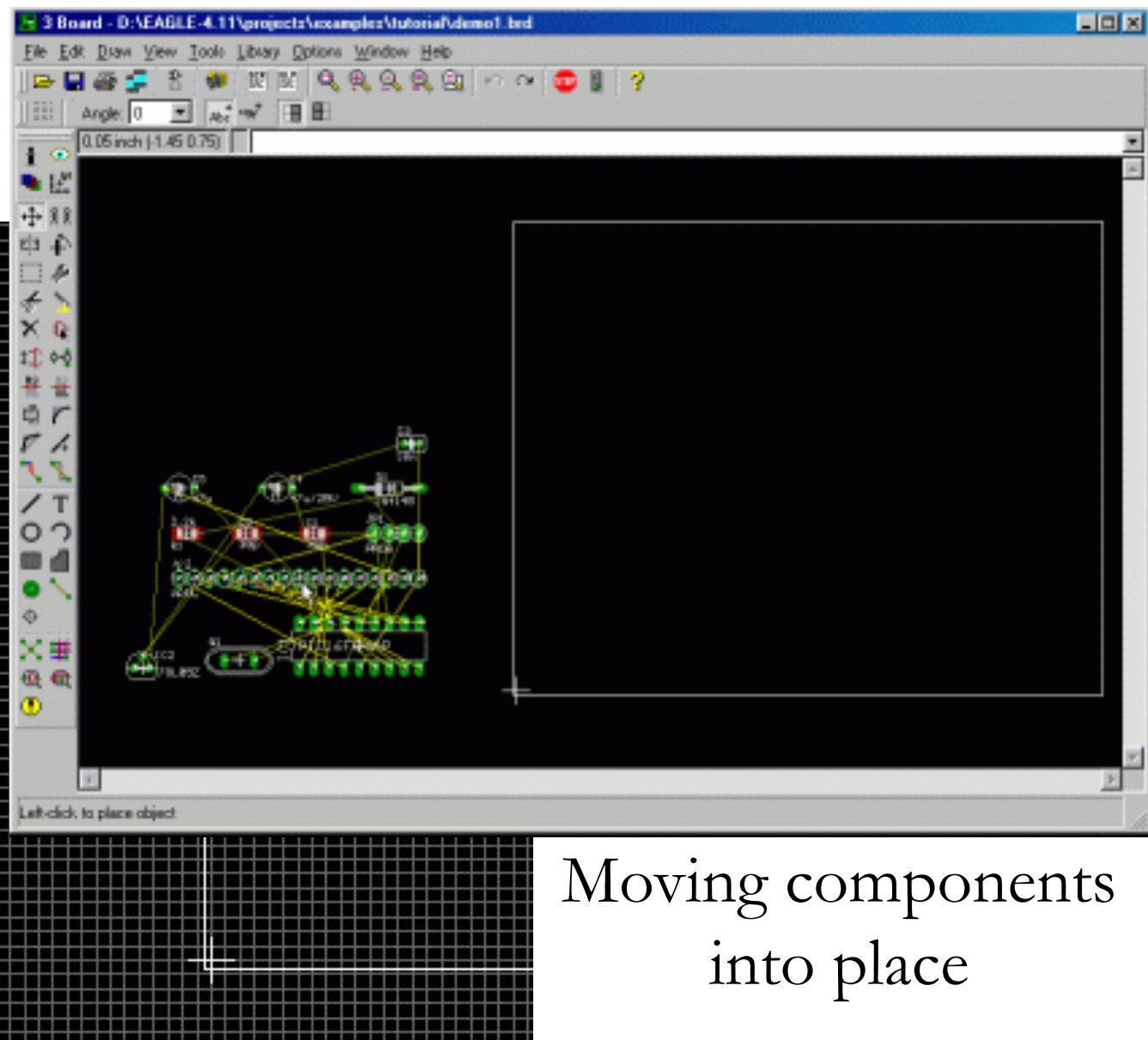
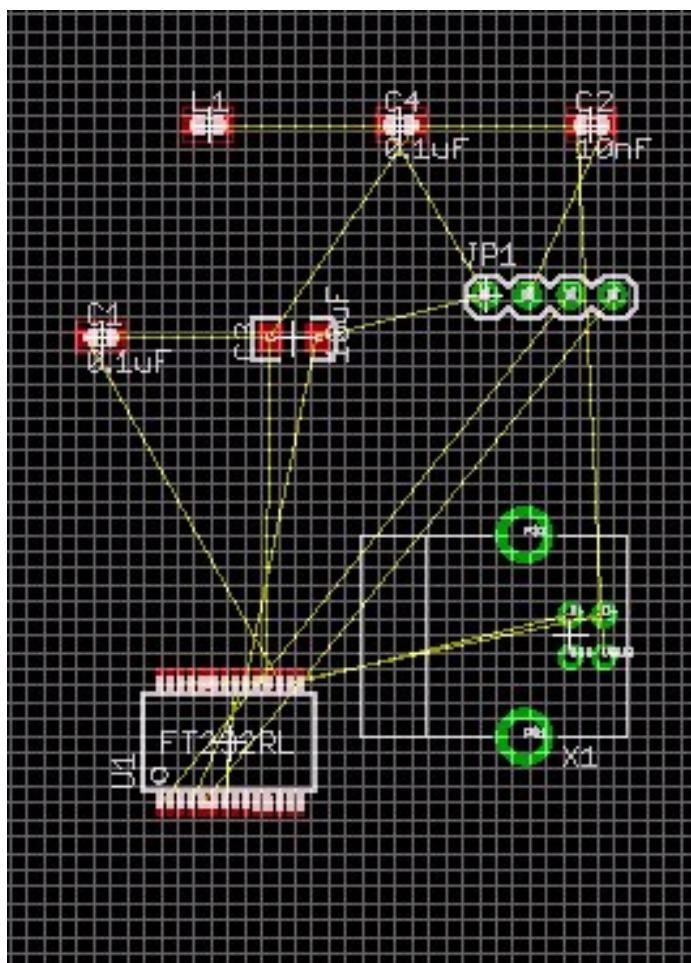
Schematic Capture

- Pin in/outs
- Components
- Interconnections
- Easily Readable
- High-Level Block Diagram



Placement of the Components on the PCB

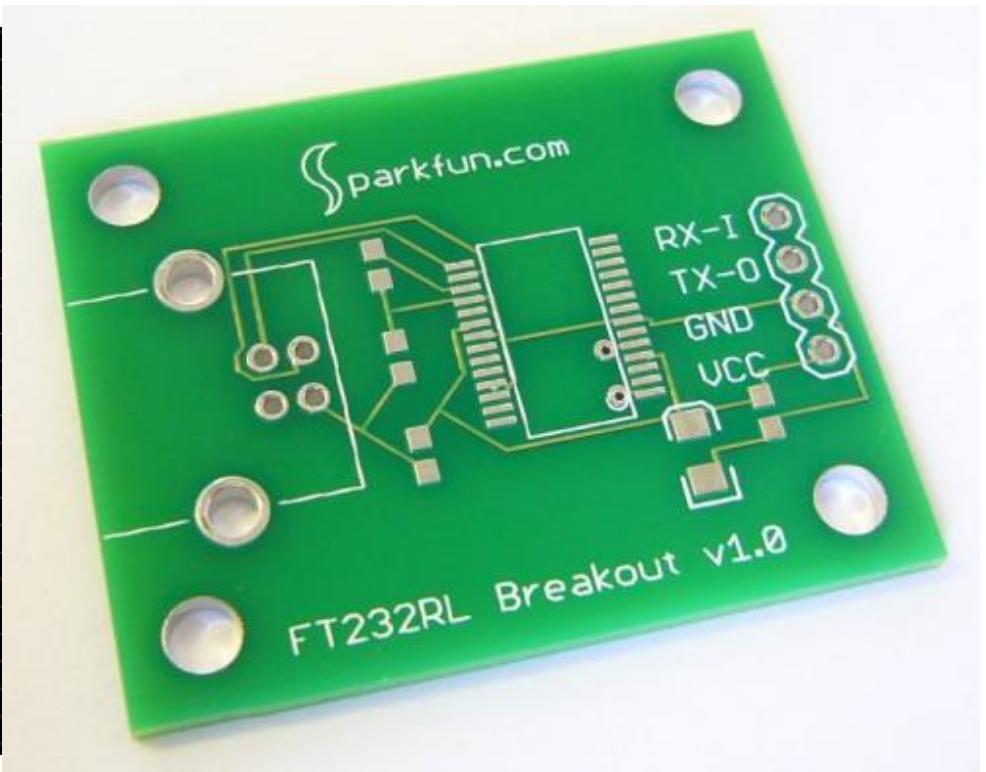
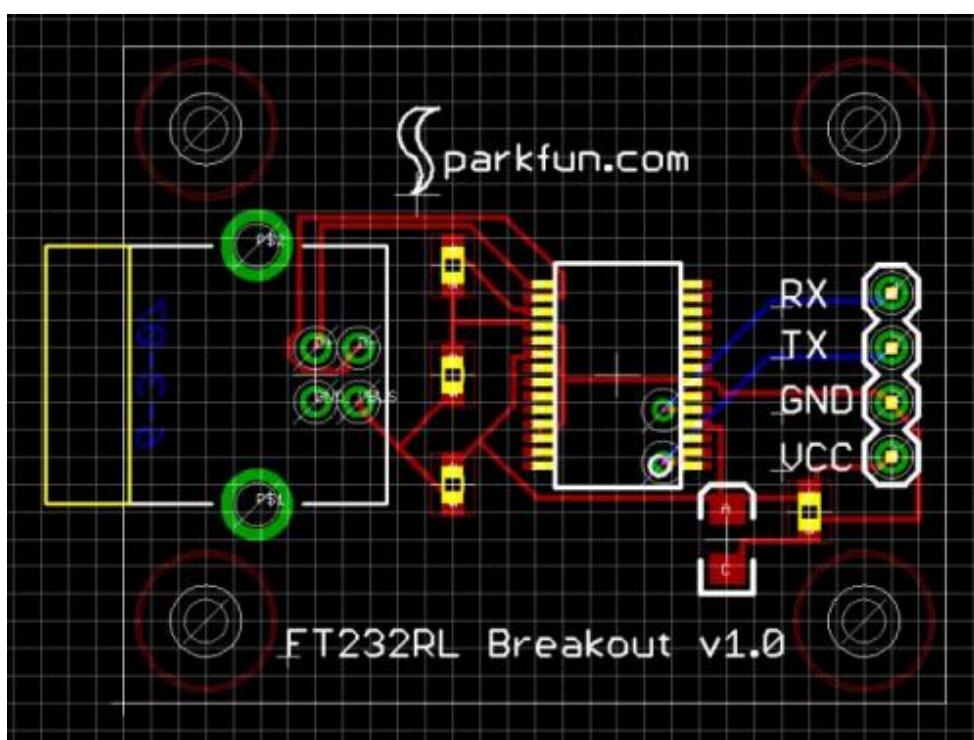
Rat's Nest
Air Wires



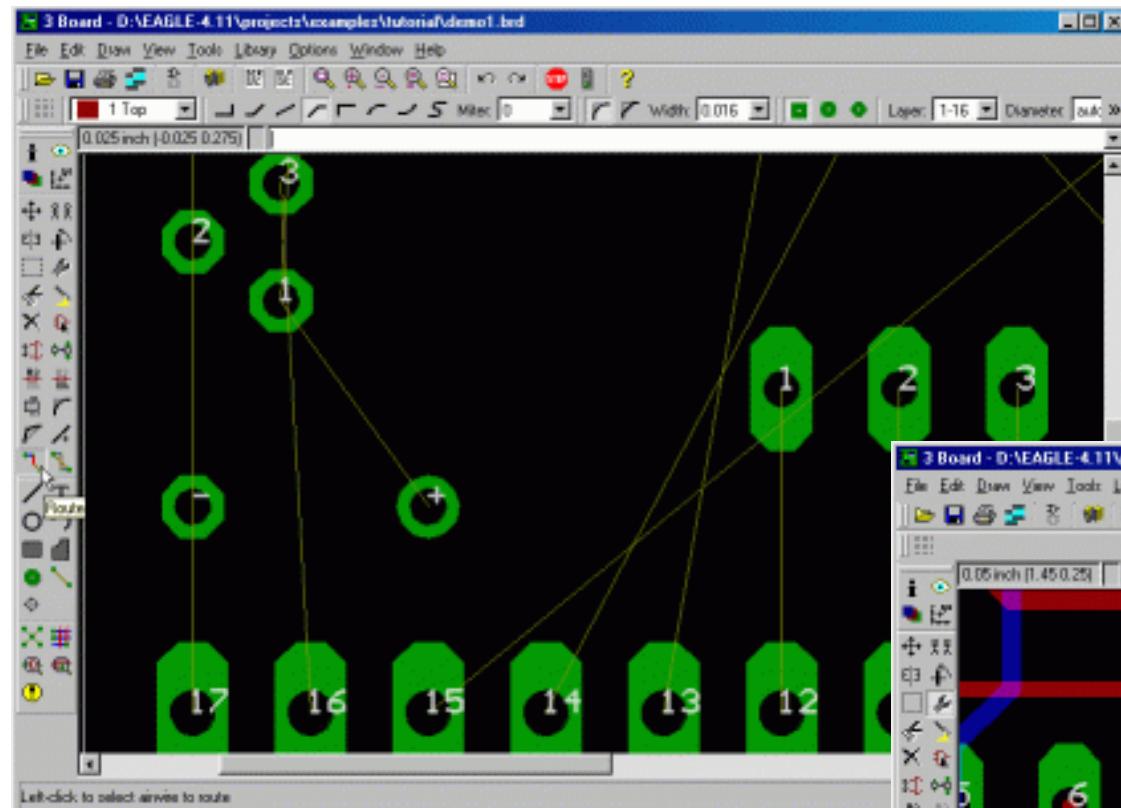
Moving components
into place

Signal Routing

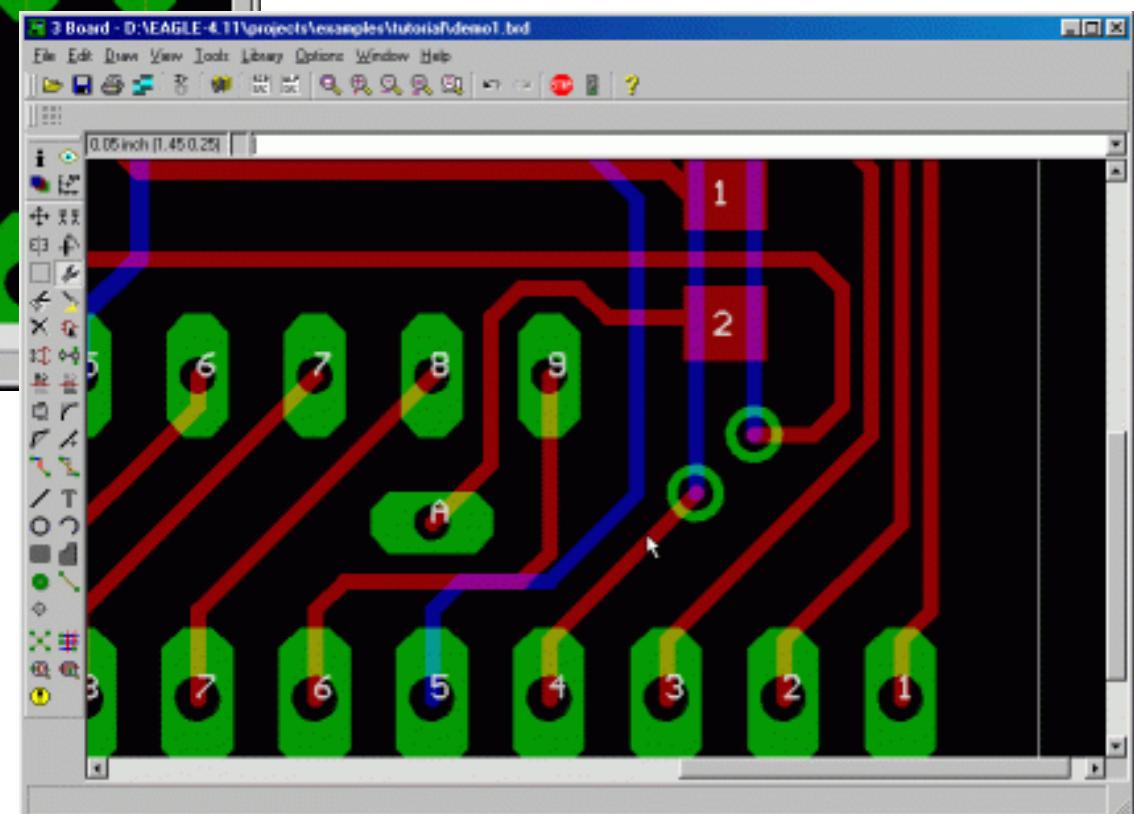
- Traces are wires connecting components
- Traces can be routed through multiple layers
- Vias are connections between layers
- Pads are copper areas for pin connections



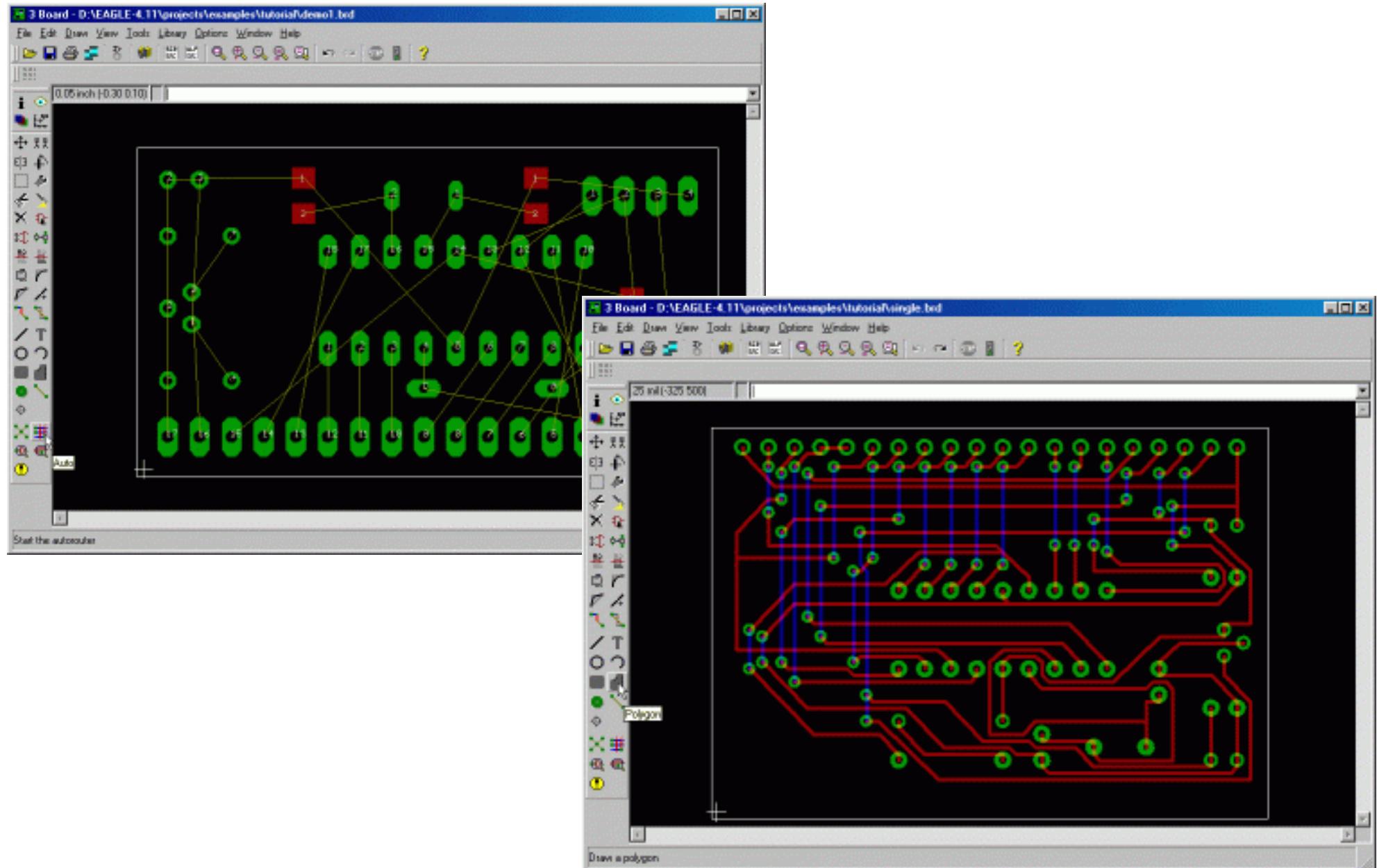
Route the Signals



Red on Top Layer
Blue on Bottom Layer

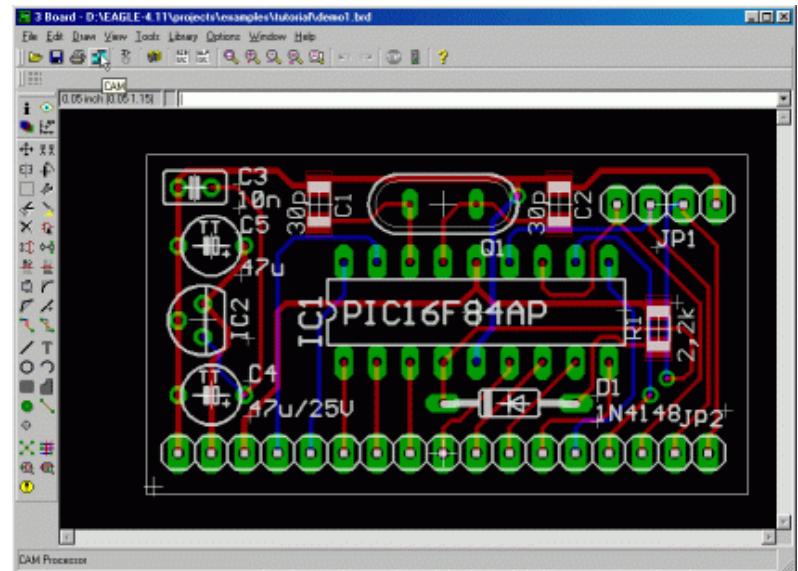


or... Auto-route the Board



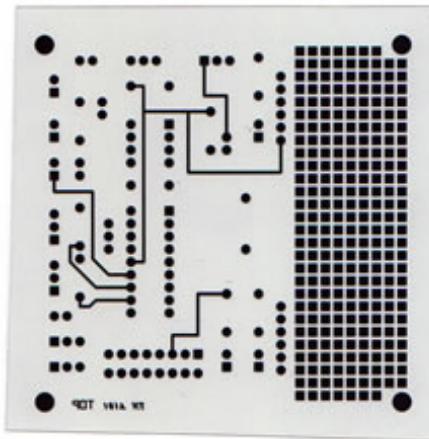
Produce Manufacturing Files

- Gerber Files
 - Industry standard file format to describe PCB layouts
 - Generated by PCB design software
 - Contains all necessary info to manufacture a circuit board (Traces, Pads, Silkscreen, etc.)
- Drill Files
 - Describes the location and size of holes
 - Zipped with Gerber files and sent to manufacturer



PCB Manufacture Process

1. Film Generation



A film is a full-size template for a pattern on the PCB. A PCB design will result in about 10 different films (patterns).

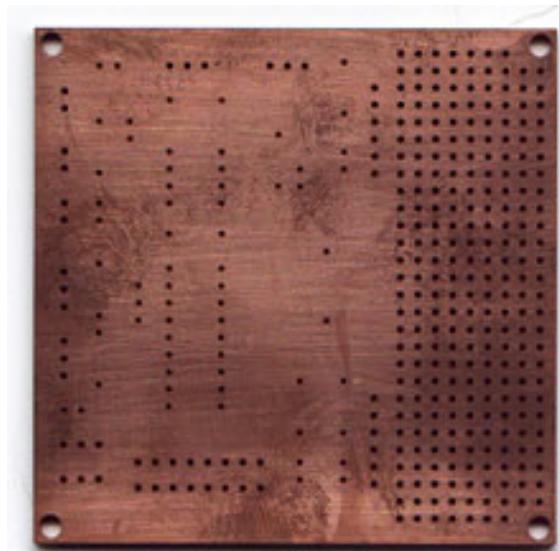
Steps in PCB Manufacture

2. Shear Raw Material



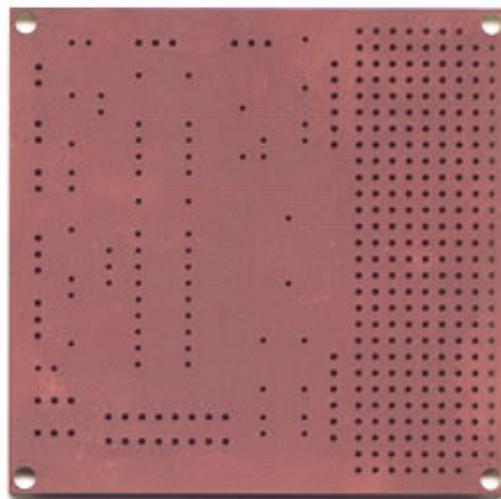
Industry standard
0.059" thick, copper
clad, two sides

3. Drill Holes



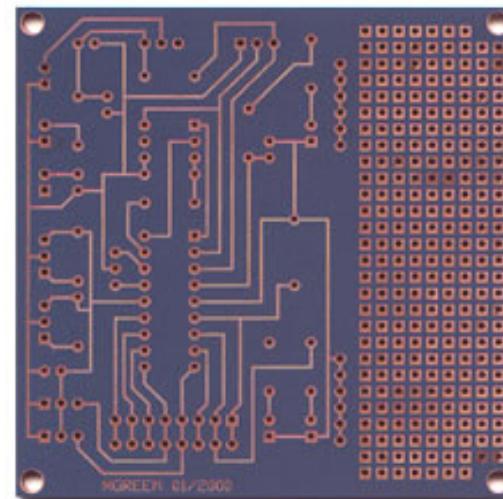
Steps in PCB design

4. Electrolus copper



Deposit copper coating inside the barrels of every hole

5. Apply Image



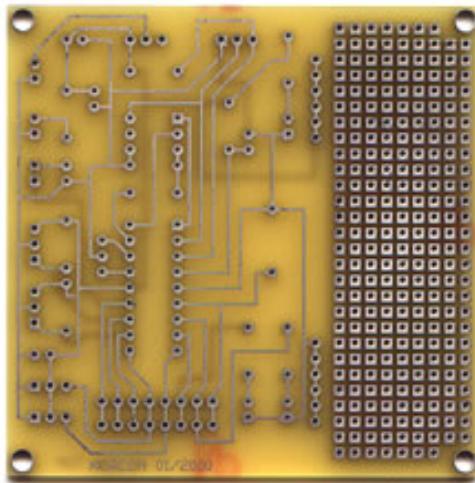
Apply photosensitive dry-film material (blue) over entire surface.

Using mask, expose and develop selected areas from dry-film.

Apply tin coating to exposed areas.

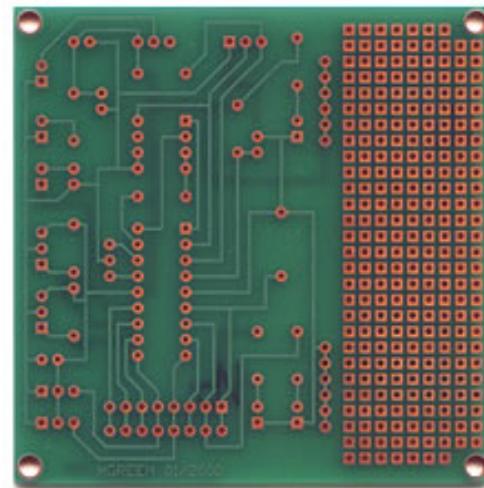
Steps in PCB Design

6. Strip and Etch



Remove dry-film
Etch exposed copper
Tin protects the copper
circuitry from being etched

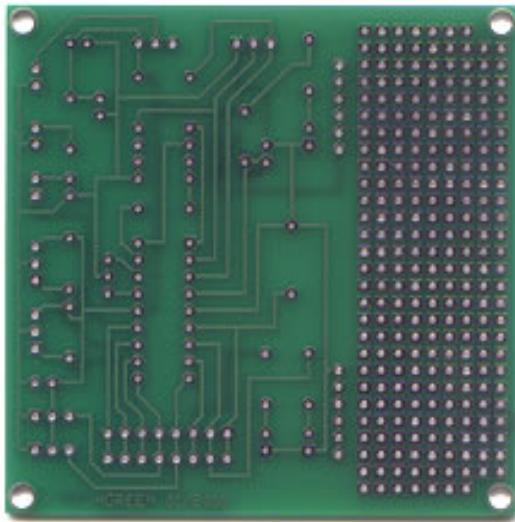
7. Solder Mask



Apply solder mask (green
varnish) to entire board
with the exception of
solder pads

Steps in PCB Design

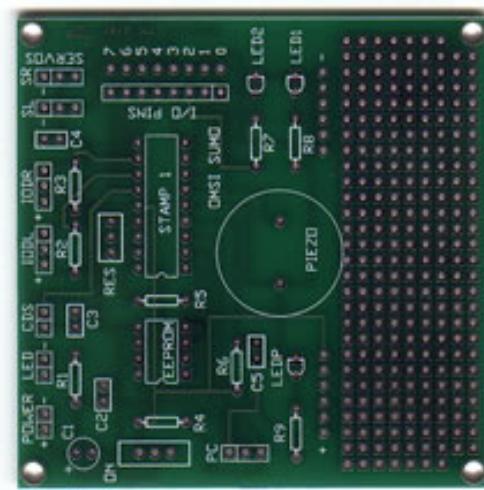
8. Solder Coat



Apply solder coat to exposed pads

Solder will not stick to
solder mask (green)

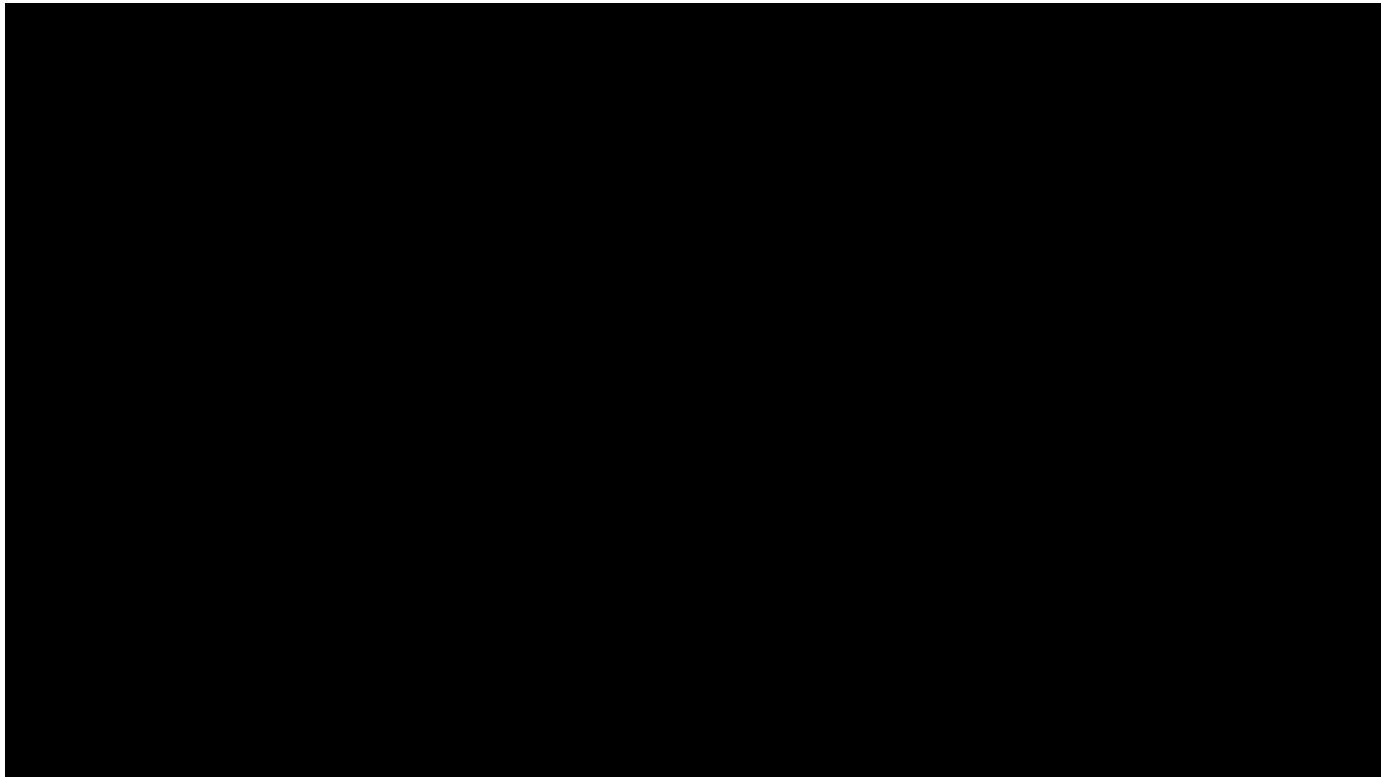
9. Silkscreen



Silk screen print white markings on board

Markings have no electrical purpose, only to help in assembly

Automated Board Assembly



Industrial board assembly

- For larger quantities, boards can be sent out to be “stuffed” with components.
- Smaller and smaller production runs are possible.
- Companies use “pick and place” machines along with components in “tape” form to speed the process.

