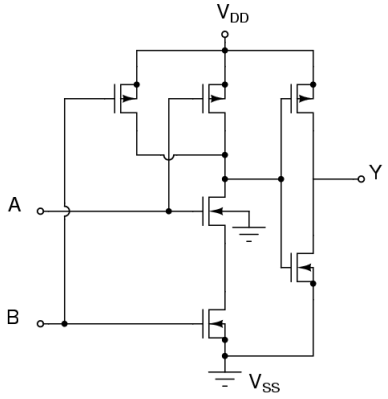
Digital Electronics

In digital circuits only two values of V_{in} or V_{out} are considered, Low (L) or High (H). The two values correspond to the logical states True (T) or False (F).

CMOS AND circuit



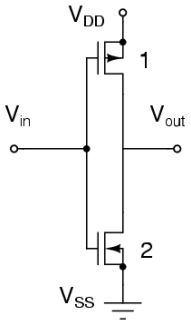
(L)ow voltage = V_{SS}
(H)igh voltage = V_{DD}

A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

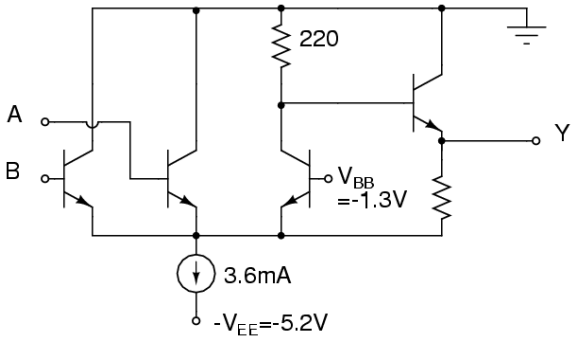
Typ. CMOS output stage

If V_{in}=5V:
V_{GS1}=0, FET_1 off
V_{GS2}=5V, FET_2 on
V_{out} = 0V

If V_{in}=0V:
V_{GS1}=-5V, FET_1 on
V_{GS2}=0V, FET_2 off
V_{out} = 5V



ECL OR circuit



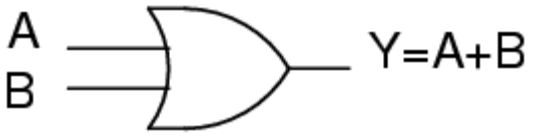
A	*B*	*Y*
L	L	L
L	H	H
H	L	H
H	H	H

Y=H if A=H or B=H

Standard convention is that L=FALSE=0 and H=TRUE=1 respectively. Logically the above circuits function as logical AND and OR gates respectively.

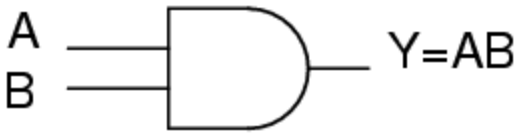
Combinational Logic Gates

OR Gate



A	B	Y
F	F	F
T	F	T
F	T	T
T	T	T

AND Gate



A	B	Y
F	F	F
T	F	F
F	T	F
T	T	T

Some Boolean Identities

A+B+C = (A+B)+C = A+(B+C)

A+B = B+A

A+A = A

A+T = T

A+F = A

ABC = (AB)C = A(BC)

AB = BA

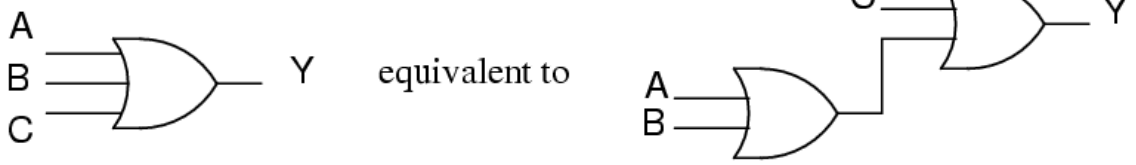
AA = A

AT = A

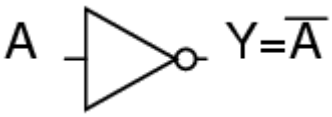
AF = F

A(B+C) = AB + AC

The first identity indicates how a logic gate having more than two inputs should behave, eg:



NOT or INVERT Gate



Truth Table

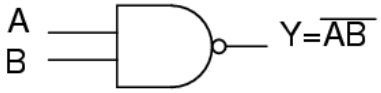
A	Y
F	T
T	F

Boolean Identities

$$\overline{\overline{A}} = A$$
$$A + \overline{A} = T$$
$$A\,\overline{A} = F$$
$$A + \overline{A}\,B = A + B$$

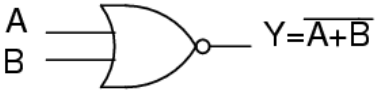
Additional gates formed by combining NOT with AND and OR

NAND

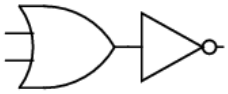


equivalent to

NOR



equivalent to



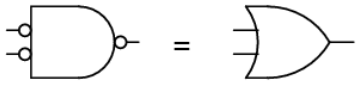
A	B	Y
F	F	T
T	F	T
F	T	T
T	T	F

A	B	Y
F	F	T
T	F	F
F	T	F
T	T	F

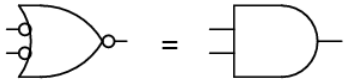
De Morgan's Theorem

If we had identified L voltage with TRUE and H with FALSE, then the NAND and NOR circuits would actually have implemented the OR and AND functions respectively. Interchanging the assignments L=F, H=T to L=T, H=F is equivalent to applying the logical complement to all entries of a truth table...

$$\overline{\overline{A} \bullet \overline{B} \bullet \overline{C}} = A + B + C$$



$$\overline{\overline{A} + \overline{B} + \overline{C}} = A \bullet B \bullet C$$



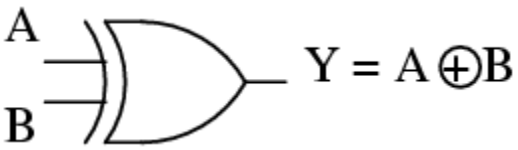
Note: a circle indicates inversion of a logic level

Implementing Logic Equations with Gates

Take the assignment between voltage levels and logic quantities to be:
L = F = 0, H = T = 1
Then De Morgan's theorem will be used to treat signals that are "active low."

General problem: Given a truth table, find some combination of logic gates that satisfy the table. The first step is to find a set of logic equations to solve the truth table. There are two ways to do this: (1) by inspection, a.k.a. 'divine intuition', (2) by Karnaugh Map. Then algebraic manipulation can be used to reduce the expression.

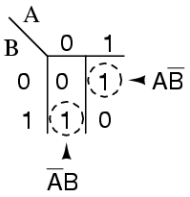
Example: XOR (exclusive OR) gate



A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

By inspection of the table and using De Morgan's theorem $Y = (A+B)\overline{(A\cdot B)} = (A+B)(\overline{A}+\overline{B})$
$= \cancel{A\overline{A}} + \cancel{B\overline{B}} + A\overline{B} + \overline{A}B$

K-Map:

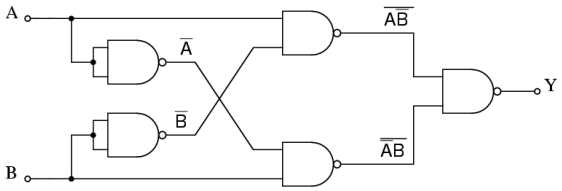


$\Rightarrow Y = A\overline{B} + \overline{A}B$

Two implementations using NAND gates $Y = A\overline{B} + \overline{A}B$

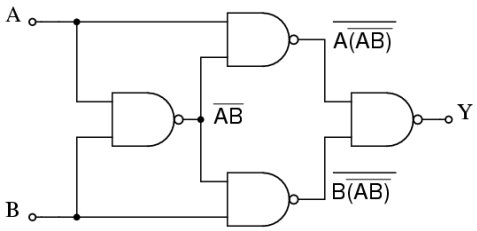
Using De Morgan's Laws:

$\overline{Y} = \overline{A\overline{B}}\cdot\overline{\overline{A}B} \Rightarrow Y = \overline{\overline{A\overline{B}}\cdot\overline{\overline{A}B}}$



Another solution using only 4 gates:

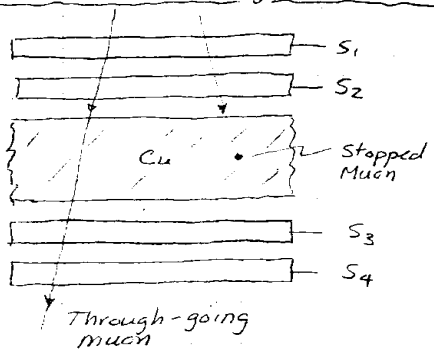
$Y = (A+B)\overline{(A\cdot B)} = (A+B)(\overline{A}+\overline{B})$
$= A\cdot\overline{(A\cdot B)} + B\overline{(A\cdot B)}$
$= \overline{\overline{A\cdot\overline{(A\cdot B)}}\cdot\overline{B\overline{(A\cdot B)}}}$



Note: all logic functions can be built up from NAND gates alone (or from NOR gates).

One of the primary uses of digital electronic circuits is to make logical decisions in an experimental situation. An example is the 319 Lab "Muon Lifetime and Magnetic Moment".

Muons created in the upper atmosphere are detected in scintillators $S_1 - S_4$ (ie particle → light pulse → photomultiplier tube → electronic pulse → comparator → Sn) sandwiched around a thick copper block. Most muons pass through the entire stack, leaving a signal in all 4 scintillators. However a few will come to rest in the copper block and these are the ones to study.



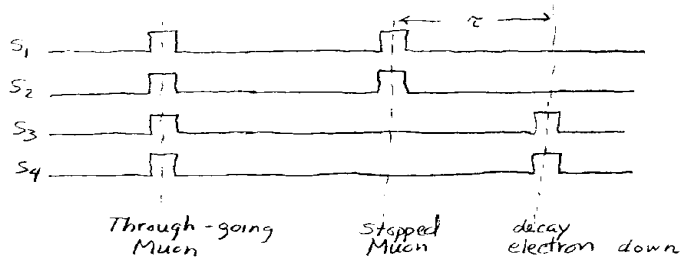
So, the logic equation describing a muon stopping in the block is

$$Mu - stop = (S_1 \cdot S_2) \cdot (\bar{S}_3 \cdot \bar{S}_4)$$

After a muon comes to rest it will decay into an electron (plus neutrino) within an average ~ 2.7 μsec. The electron could be detected in this apparatus either going up or down; the logic equations would be.

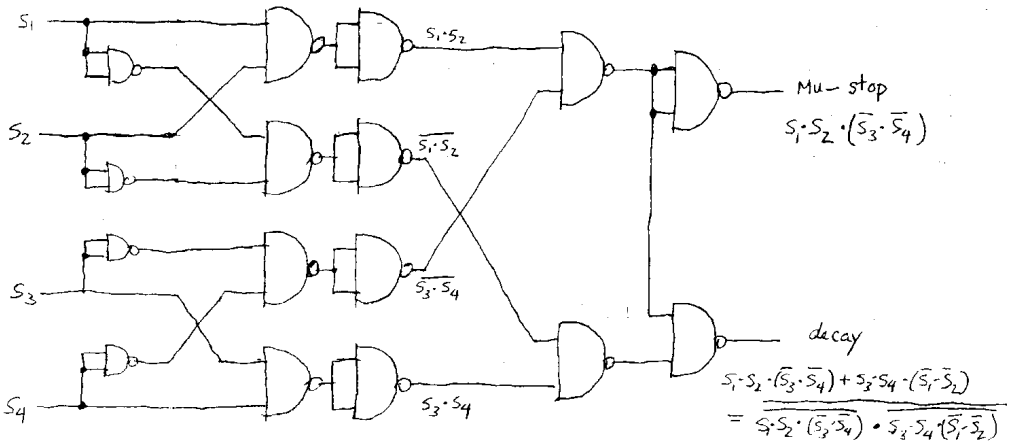
$$E - up = (S_1 \cdot S_2) \cdot (\bar{S}_3 \cdot \bar{S}_4) \quad \text{and} \quad E - down = (S_3 \cdot S_4) \cdot (\bar{S}_1 \cdot \bar{S}_2)$$

On a 4-trace oscilloscope . . . . .

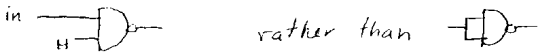


The signal of a decay is
Decay = E_up + E_down

An implementation of this logic using 2-input NAND gates is

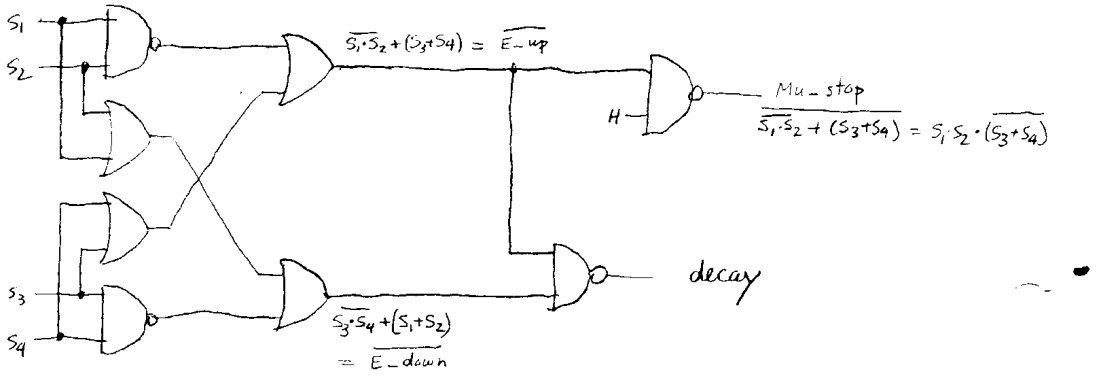


Note it is actually better for noise immunity to implement the inverter function by



This example illustrates the general result that any logic eqn can be realized by NAND gates alone (although many may be needed!)

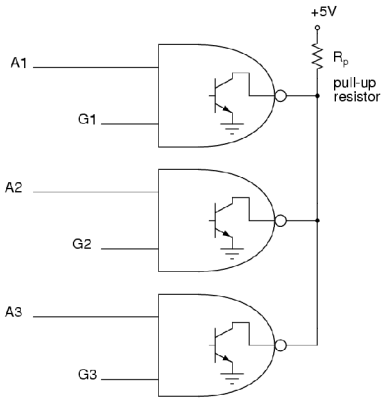
If allow the use of other 2-input gates a more compact circuit could be achieved eg suppose OR gates are available



Open collector and Tri-State outputs

In addition to LOW and HIGH output voltage levels, some gates have a third, High-Z, output state. This allows many outputs to be connected together on a single wire (or bus). Two varieties re commonly used:

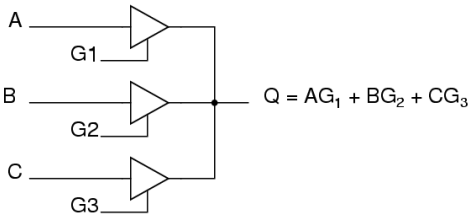
1) Open Collector



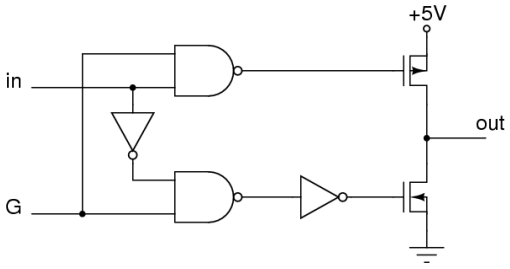
$$Q = \overline{A \cdot G_1 + B \cdot G_2 + C \cdot G_3}$$

Disadvantages:
1) not an active pull-up $t_{rise} \sim R_p C$
2) inverted logic must be used

2) Tri-State



When both FETs are off (G=0) output is in High-Z state



Relation to Number Codes

(11)

In base-2 a number is represented by a sequence of bits (= binary digits) each assuming one of two possible values 0 or 1. Logic circuits can be used to do binary-arithmetic operations if we identify the logic states L, H with the two possible bit values 0, 1.

The representation is similar to base-10, ie there is a digit multiplying each power of the base number

MSB ↓ LSB ↓

eg. $1\ 1\ 0\ 1_2 = 1\times 2^0 + 0\times 2^1 + 1\times 2^2 + 1\times 2^3 = 13_{10}$

MSB ≡ most significant bit
LSB ≡ least " "

Converting from decimal is accomplished by repeated div. by 2

13/2	= 6	rem 1	— LSB	
6/2	= 3	rem 0		i.e $13_{10} = 1101$
3/2	= 1	rem 1		
1/2	= 0	rem 1	— MSB	

Arithmetic with unsigned (ie positive only) numbers proceeds just as the base-10 case

$$\begin{array}{r} 2 \\ +\ 7 \\ \hline 9 \end{array} \qquad \begin{array}{r} 0\ 0\ 1\ 0 \\ 0\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 1 \end{array} = 9_{10}$$

$$\begin{array}{r} 2 \\ \times\ 7 \\ \hline 14 \end{array} \qquad \begin{array}{r} 0\ 0\ 1\ 0 \\ 0\ 1\ 1\ 1 \\ \hline 0\ 0\ 1\ 0 \\ +\ 0\ 0\ 1\ 0 \\ +\ 0\ 0\ 1\ 0 \\ \hline 1\ 1\ 1\ 0 \end{array} = 14_{10}$$

An n-bit sequence has 2^n possible values. Thus it can represent the positive integers $0, 1, \ldots 2^n - 1$. There are two popular mappings from bit seq. to positive integers;

i) so-called "natural" binary - already used in previous example.

ii) Grey code - bit patterns corresponding to successive numbers differ by only one bit. This code is useful in noisy environments.

An n-bit sequence can also be used to represent a range of neg and positive numbers $-2^{n-1}, -2^{n-1}+1, \ldots, -1, 0, 1, \ldots 2^{n-1}-1$.

The three most popular codes for representing signed integers are illustrated in the table (taken from H & H pg 477).

In the sign/magnitude representation the n^{th} bit is used a sign bit with the others representing the magnitude in "natural" binary - Its not useful for computation

TABLE 8.1. 4-BIT SIGNED INTEGERS IN THREE SYSTEMS OF REPRESENTATION

Integer	Sign-magnitude	Offset binary	2's comp
+7	0111	1111	0111
+6	0110	1110	0110
+5	0101	1101	0101
+4	0100	1100	0100
+3	0011	1011	0011
+2	0010	1010	0010
+1	0001	1001	0001
0	0000	1000	0000
–1	1001	0111	1111
–2	1010	0110	1110
–3	1011	0101	1101
–4	1100	0100	1100
–5	1101	0011	1011
–6	1110	0010	1010
–7	1111	0001	1001
–8	–	0000	1000
(–0)	1000	–	–

Both Offset Binary and 2's complement Binary have the useful feature that the ordering (in the sense of "natural" binary) is the same for both positive and negative integers.

Offset-Binary is the natural code to use for Analog to Digital converters (ADC) and Digital to Analog converters (DAC).

2's-complement binary has the advantage that the same hardware can be used for addition and subtraction, which is just addition of a negative number.

(13)

Binary Subtraction

To see why the 2's complement representation allows the same hardware to be used for addition and subtraction consider the subtraction of two 4-bit numbers, B minus A.

The property of "minus A" should be that A minus A = 0.
Note that since the INVERT function converts $0 \to 1$ & $1 \to 0$.

$$A \text{ plus } \bar{A} = 1111$$

$$\Rightarrow \quad A \text{ plus } \bar{A} \text{ plus } 1 = 10000$$

Since we are working with only 4-bit numbers the 5th (MSB) must be ignored.

$$\Rightarrow \quad 10000 \text{ minus } A = \underbrace{\bar{A} \text{ plus } 1}_{\text{2's complement.}}$$

So finally $\boxed{B \text{ minus } A = B \text{ plus } (\bar{A} \text{ plus } 1)}$

Note this computation will produce a 5th bit which must be discarded.

Gate Level Implementation of an Adder

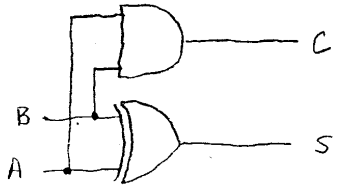
Consider first just two bits A and B. The output includes both a sum bit S and a carry C, in case A=1 and B=1.
Start by writing the truth table

in		out	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

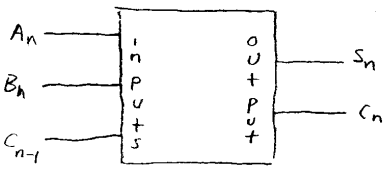
By inspection one can easily see

$S = A \oplus B$ (XOR) and $C = A \cdot B$

$\Rightarrow$



This is not a complete adder - in fact it's called a Half Adder. A Full-Adder would include a carry-in from a previous stage as an additional input. N such stages could be connected to make an n-bit adder.



Full Adder

Truth Table

A_n	B_n	C_{n-1}	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Logic eqn follows from the combinations of all inputs yielding an output = 1.

$$S_n = \bar{A}_n \cdot \bar{B}_n \cdot C_{n-1} + \bar{A}_n \cdot B_n \cdot \bar{C}_{n-1} + A_n \cdot \bar{B}_n \cdot \bar{C}_{n-1} + A_n \cdot B_n \cdot C_{n-1}$$

$$C_n = \bar{A}_n \cdot B_n \cdot C_{n-1} + A_n \cdot \bar{B}_n \cdot C_{n-1} + A_n \cdot B_n \cdot \bar{C}_{n-1} + A_n \cdot B_n \cdot C_{n-1}$$

Each term which involves a product of all inputs is called a minterm. The expressions for S_n and C_n are a sum (OR) of minterms - such a form is called standard or canonical form.

To reduce these expressions to forms amenable to 2-input gates use Boolean algebra.

$A_n \oplus B_n$ - XOR

eg. $S_n = \underbrace{(\bar{A}_n \cdot \bar{B}_n + A_n \cdot B_n)}_{\overline{(A_n + B_n)} + \overline{(\bar{A}_n + \bar{B}_n)}} \cdot C_{n-1} + \bar{C}_{n-1} \cdot \underbrace{(\bar{A}_n \cdot B_n + A_n \cdot \bar{B}_n)}$

$$\overline{(A_n + B_n) \cdot (\bar{A}_n + \bar{B}_n)} = \overline{(A_n \cdot \bar{B}_n + \bar{A}_n \cdot B_n + \bar{A}_n A_n + \bar{B}_n \cdot B_n)}$$

$$\Rightarrow \quad S_n = C_{n-1} \cdot \overline{(A_n \oplus B_n)} + \bar{C}_{n-1} \cdot (A_n \oplus B_n) = C_{n-1} \oplus (A_n \oplus B_n)$$

The motivation for manipulating the terms to look like XORs came from the observation of an XOR in the Half Adder circuit.

By inspection of the Full-Adder's Truth Table, a carry-out C_n occurs when two or more of the inputs = 1. This could be described by,

$$C_n = B_n \cdot C_{n-1} + C_{n-1} \cdot A_n + A_n \cdot B_n + A_n \cdot B_n \cdot C_{n-1} .$$

(ignore since it's redundant w/ previous 3 terms.)

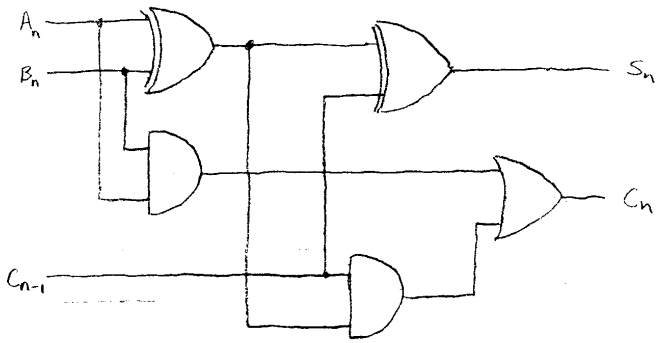
Eq^n would require 5 more gates.....

However working directly from the previous eq^n a more compact implementation can be achieved,

$$C_n = \bar{A}_n \cdot B_n \cdot C_{n-1} + A_n \cdot \bar{B}_n C_{n-1} + A_n \cdot B_n \cdot (\bar{C}_{n-1} + C_n)$$

$$= (\bar{A}_n \cdot B_n + A_n \cdot \bar{B}_n) \cdot C_{n-1} + A_n \cdot B_n$$

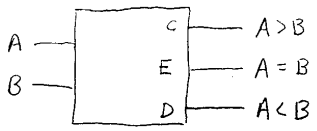
$$= \underbrace{(A_n \oplus B_n)}_{\text{already exists from } S_n} \cdot C_{n-1} + A_n \cdot B_n$$



Note the implementation is not unique. eg. the student manual uses 2 ANDs & 2 ORs to form C_n. If one were required to use only AND & OR the implementation would have been even more different.

(17)

Ex. Binary Magnitude Comparator



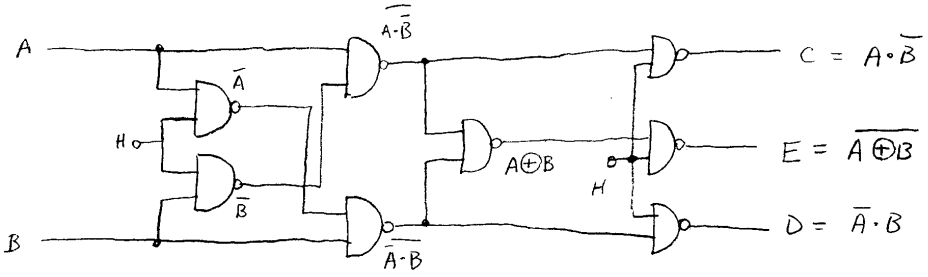
First consider comparing single bits.

An equality is detected with an exclusive NOR, $\overline{XOR}$

$$E = \overline{A\cdot\bar{B} + \bar{A}\cdot B} = \begin{cases} 1 & \text{if } A=B \\ 0 & \text{if } A\neq B \end{cases}$$

The $A>B$ condition is given by $C = A\cdot\bar{B}$
and $A<B$ " " " $D = \bar{A}\cdot B$

One implementation using NAND gates is



Now consider a 4-bit comparator.

- Equality requires $A_0 = B_0$, $A_1 = B_1$ ... $A_3 = B_3$. Therefore

$$E = E_0\cdot E_1\cdot E_2\cdot E_3$$

where each bit-wise equality is provided by an $\overline{XOR}$ as above.

- The inequality $A>B$ requires

$A_3 > B_3$

or $A_3 = B_3$ and $A_2 > B_2$

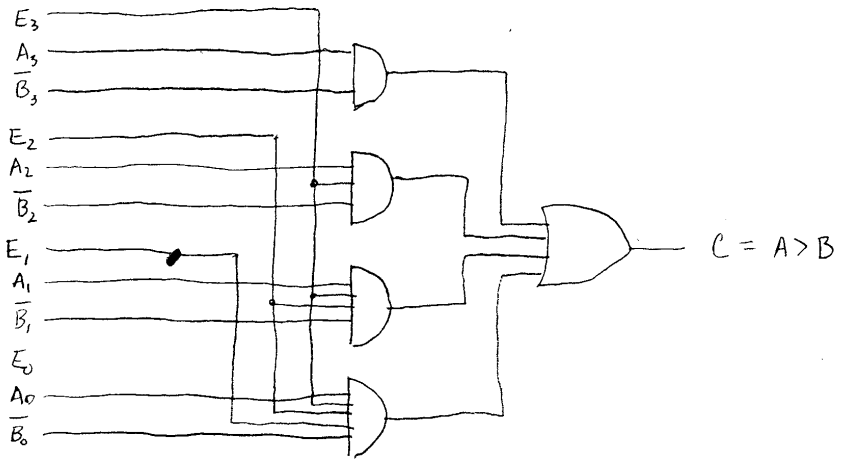
or $A_3 = B_3$ and $A_2 = B_2$ and $A_1 > B_1$

or $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 > B_0$

(18)

So, the logic equation for $A > B$ is

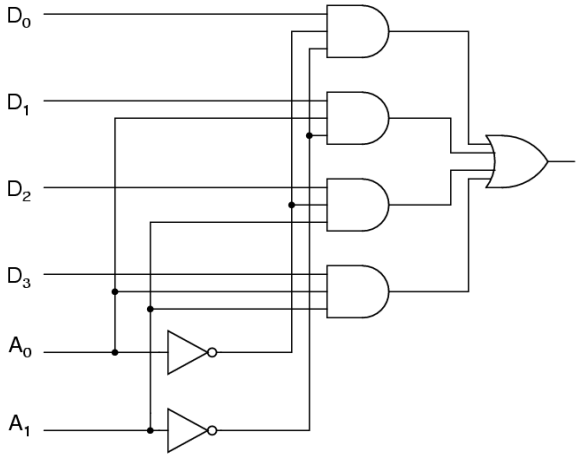
$$C = A_3 \cdot \bar{B}_3 + E_3 \cdot A_2 \cdot \bar{B}_2 + E_3 \cdot E_2 \cdot A_1 \cdot \bar{B}_1 + E_3 \cdot E_2 \cdot E_1 \cdot A_0 \cdot \bar{B}_0$$



The E_n come from circuits of the previous page.

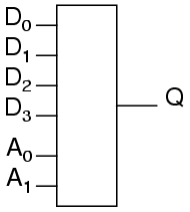
Multiplexers and Decoders

4-to-1 multiplexer (fan-in)



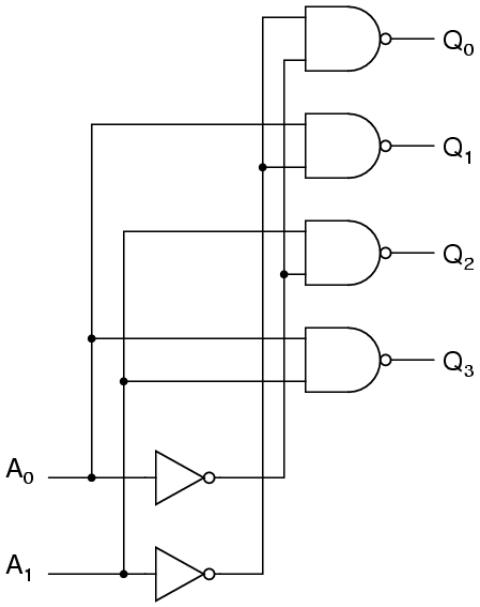
Address A[1:0] selects output from D[3:0]

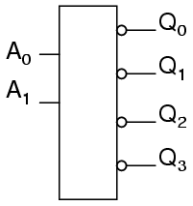
Abbreviated symbol



A_0	A_1	Q
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

2-to-4 decoder (fan-out)





A_0	A_1	Q_0	Q_1	Q_2	Q_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Address A[1:0] clears one bit from Q[3:0]

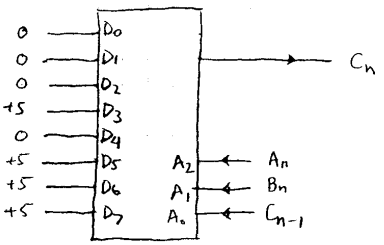
Both can be used to implement arbitrary functions of two boolean variables (bits)
Often these devices have $\overline{\text{enable}}$ inputs for tristate outputs.

Implementing Arbitrary Logic Eq^{n}s w/ a Multiplexer

(19)

"The quick & dirty solution"... An example will illustrate the technique. Suppose you wanted to implement the carry-out logic of the Full Adder without going through any logic minimization. => Apply inputs to address line of a multiplexer and set D lines high or low corresponding to the line of the truth table

line #	A_n	B_n	C_{n-1}	C_n
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



Karnaugh Maps

The Karnaugh map or K-map offers a convenient way to visualize logic equations from a truth table. Consider the following logic problem:

We have a chocolate-covered coffee bean machine, that returns a bean for a 20 cent fee. The machine accepts only quarters and dimes. To make matters simple, we'll ignore the job of making change. To enter the fee, assume the machine has a drawer w/ three depressions, one for a quarter and two for dimes (like many coin-op washing machines).

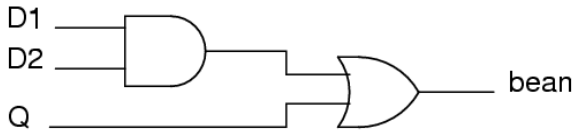
Obviously, a single quarter or two dimes will satisfy the fee requirement to get a bean. The truth table (for all possible input combinations) is:

Q	D1	D2	bean
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

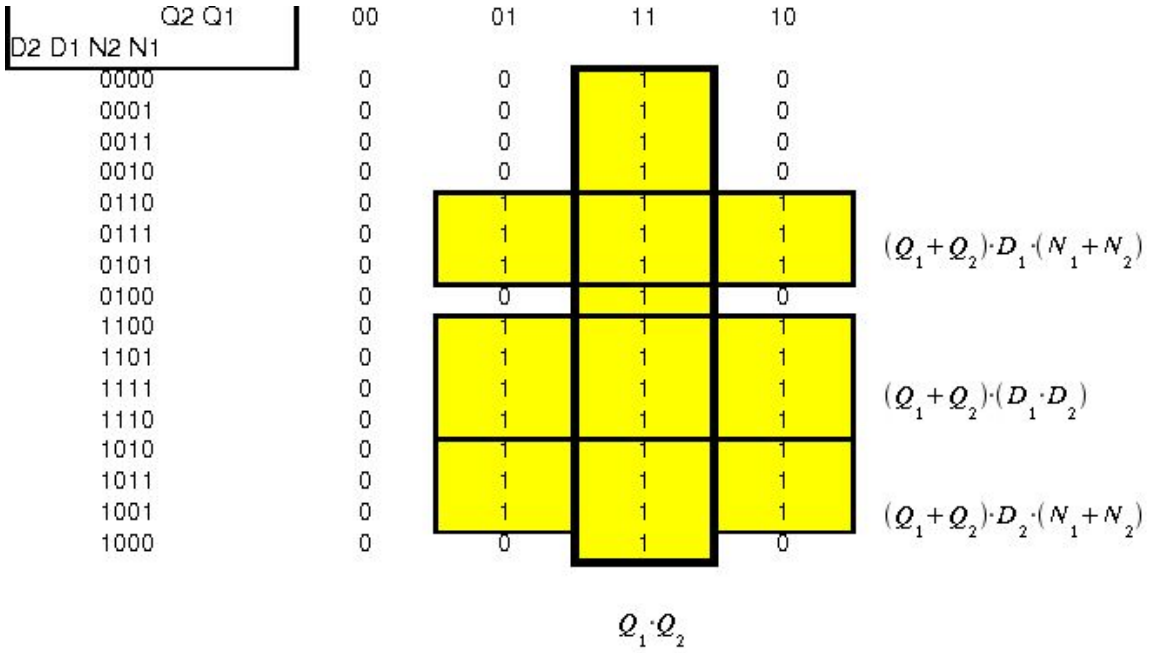
K-map for bean machine:
(note: grey code must used in all row/ columns)

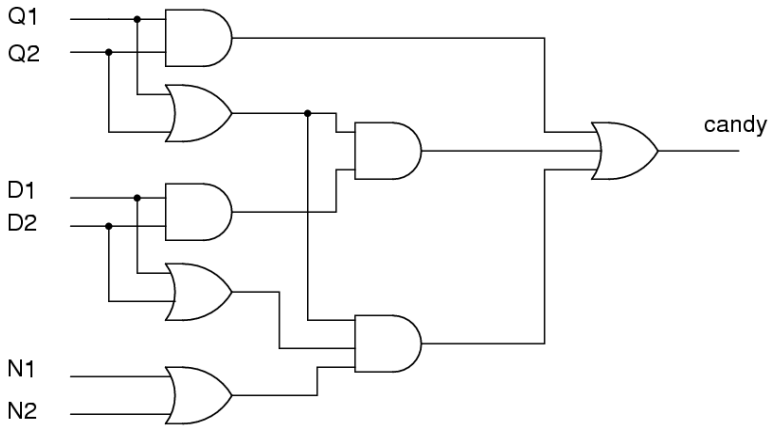
$Q \setminus D_1 D_2$	00	01	11	10	
0	0	0	1	0	
1	1	1	1	1	=Q
			$=D_1 \cdot D_2$		

Therefore: $bean = Q + D_1 \cdot D_2$



A slightly more detailed example.
Now we have a vending machine that takes up to 6 coins (2Q, 2D, 2N). Candy may be returned if at least 40 cents is input, gum may be returned if at least 30 cents is input. Let's look at the Kmap for candy:





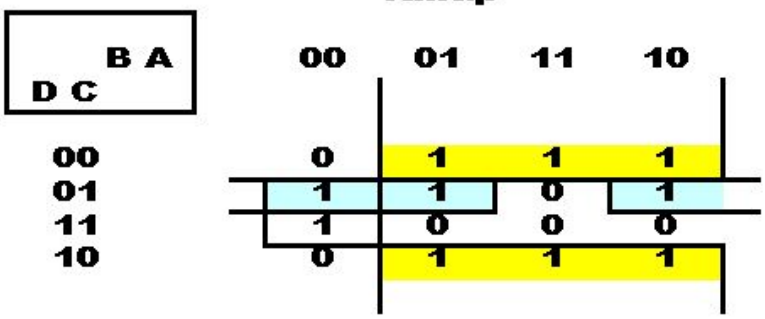
$$\text{candy} = Q_1\cdot Q_2 + (Q_1+Q_2)\cdot(D_1\cdot D_2) + (Q_1+Q_2)\cdot(D_1+D_2)\cdot(N_1+N_2)$$

More K-Map Examples:

Truth Table

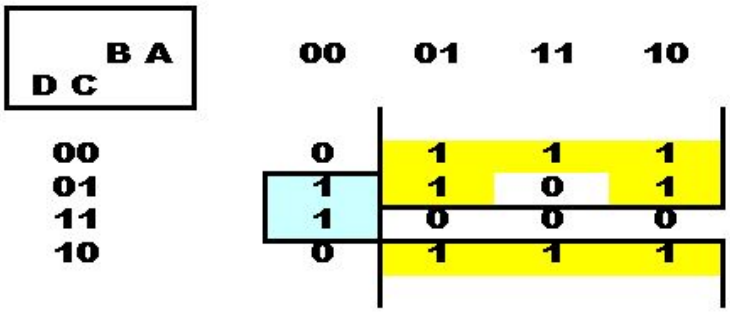
A B C D	Y
0000	0
0001	1
0010	0
0011	1
0100	1
0101	1
0110	1
0111	0
1000	1
1001	1
1010	1
1011	0
1100	1
1101	0
1110	1
1111	0

Kmap



Soultion 1:

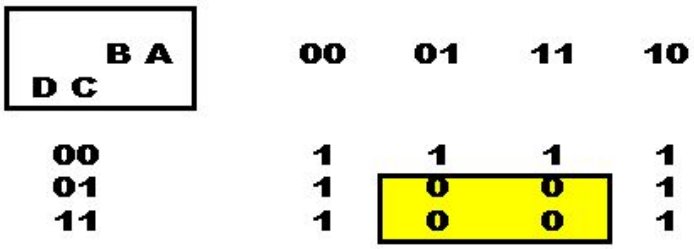
$$Y=(A+B)\cdot\overline{C}+\overline{C}\cdot\overline{D}\,\overline{(A\cdot B)}+C\cdot\overline{A}\,\overline{B}$$



Alternate solution (choose large region and remove "hole")

$$Y=(A+B)\overline{(C\cdot D)}\cdot\overline{(A\cdot B\cdot C)}+C\cdot\overline{A}\cdot\overline{B}$$

Another Example



Can also group 0's and invert to get solution: $Y=\overline{C\cdot A}$

Grouping largest possible regions always simplifies the problem

(20)

Logic Race Conditions

This describes a transient (~10ns) wrong output state.

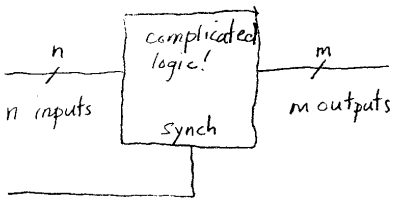
Occurs because

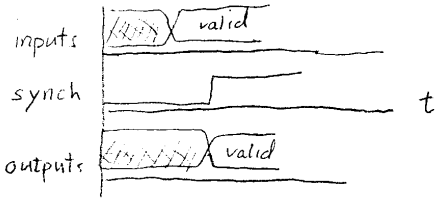
i) internal propagation delays may not be matched for different inputs

ii) input signals may not be applied simultaneously (precisely)

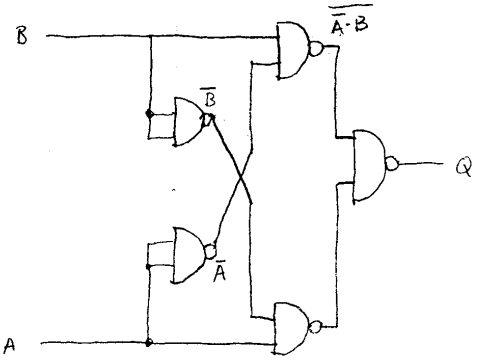
* iii) logic circuit outputs respond continuously in time according to inputs.

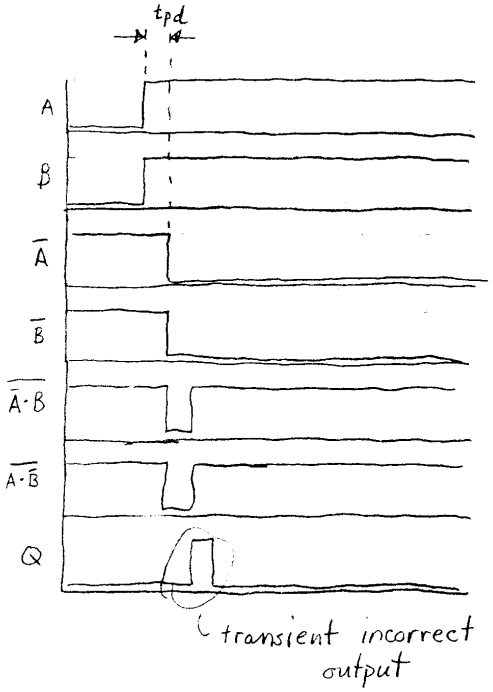
Although one can't do much about items i) & ii), by designing circuits whose outputs change only when a synchronization signal is applied one can avoid Logic race condition. This type of logic is called sequential logic - described later.





Example of logic race in XOR





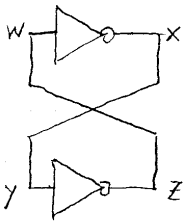
Even though gate propagation delays might be matched, the extra gate delay needed for $\bar{A}$, $\bar{B}$ causes glitch at output.

Sequential Logic

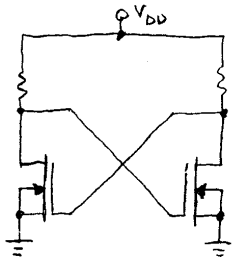
21)

In combinational logic circuits the outputs depend only on the present (or τ_{pd} earlier) inputs. It is convenient to be able to store the outputs of a combinational circuit for use later in succeeding steps of some complicated algorithm. Logic circuits having memory are called sequential circuits. In general their outputs depend on previous outputs as well as perhaps present inputs. Past & present are sometimes delineated by a synchronizing, or clock, signal.

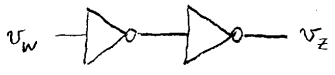
The most basic memory element is a <u>bistable latch</u> made from two inverters

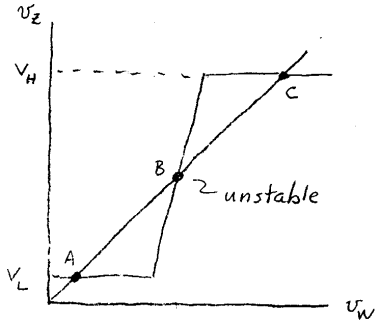


A realization using n-channel MOSFETs



By breaking the Z-W feedback, but using the constraint $v_W = v_Z$ one can see that there are 3 equilibrium points, two of which are stable (A & C) and one unstable (B)





Consider the unstable point B. Refer incremental voltages v_w etc about that point. In any circuit there is unavoidable noise

which in this case appears as an initial value $v_w(0) \neq 0$. This disturbance is amplified and after a propagation-delay-time τ appears at the output

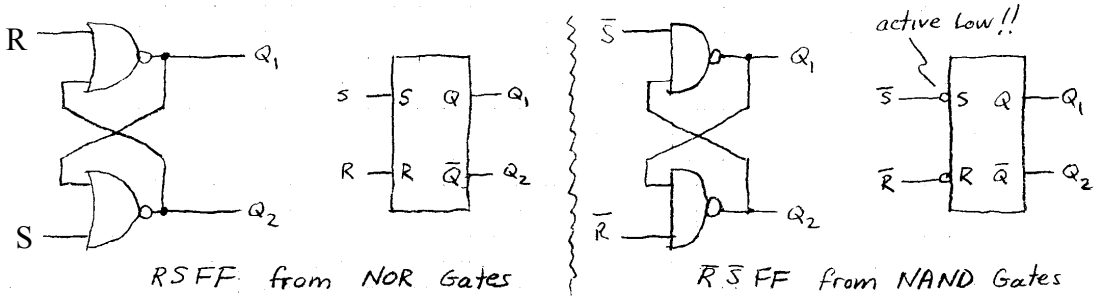
$$v_z(t) = G\, v_w(t-\tau)$$

Now when the feedback is re-established $v_z(t) = G\, v_z(t-\tau)$

$$\therefore \quad v_z(t) \cong v_w(0)\, e^{+G\frac{t}{\tau}}$$

This result, that the operating point rapidly moves to either points A or C relies only on $G \gg 1$ and τ small which is the usual case. The stability of points A and C follow because the incremental gain $G \to 0$. {i.e. $\dot{v}_z \simeq (G-1)\, v_z$ & $G-1 < 0$}

To make the bistable circuit useful we need a way to force into one of the stable states. The simplest realization of this is the Set/Reset Flip Flop (RSFF).



RSFF Truth Table

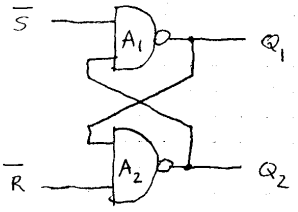
S	R	Q_1	Q_2	
0	0	0	1	"MEM" previous values held.
0	0	1	0	"MEM" previous values held.
0	1	0	1	"RESET"
1	0	1	0	"SET"
1	1	0	0	* avoid!

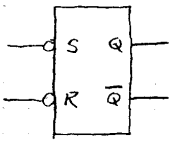
The input condition $(SR) = (1,1)$ should be avoided since upon release of S & R the resulting state SET or RESET will be unpredictable

Note that in normal operation $Q_2 = \bar{Q}_1$ in the MEM state

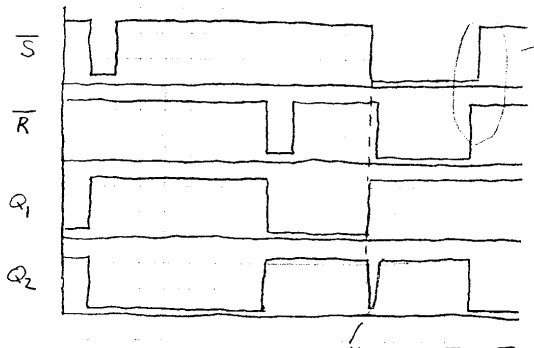
(23)

An illustration of the unpredictability of the SR asserted input state using the $\bar{S}\bar{R}$ FF built from NANDs.





$\bar{S}$	$\bar{R}$	Q_1	Q_2	
1	1	0	1	} MEM
1	1	1	0	} MEM
0	1	1	0	SET
1	0	0	1	RESET
0	0	1	1	

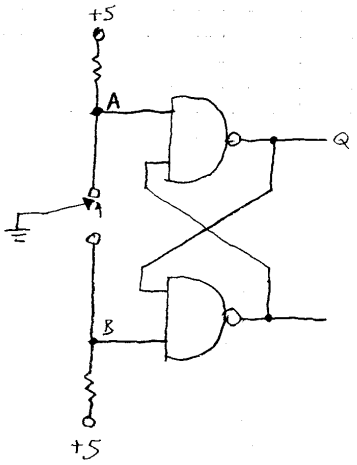


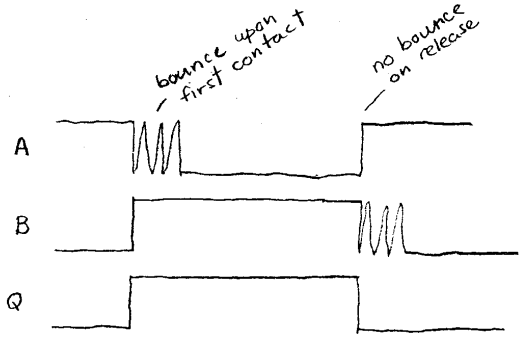
whichever is released later determines the final state of Q_1 & Q_2. Even if $\bar{S}$ & $\bar{R}$ are released simultaneously the slower of the gates A_1 & A_2 will determine the final state.

glitch if $\bar{S}$ & $\bar{R}$ don't go low simultaneously

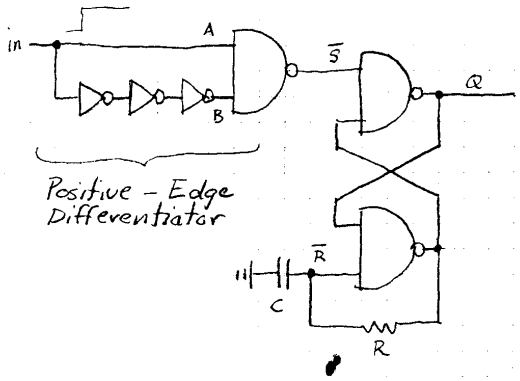
The RSFF is the basis of more complicated and useful Flip Flops. Nevertheless the simple two gate circuit has some applications

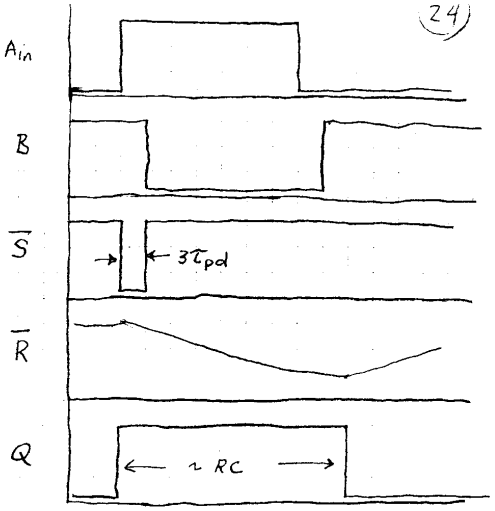
1) Switch Debouncer



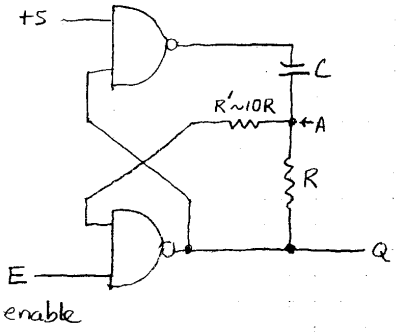


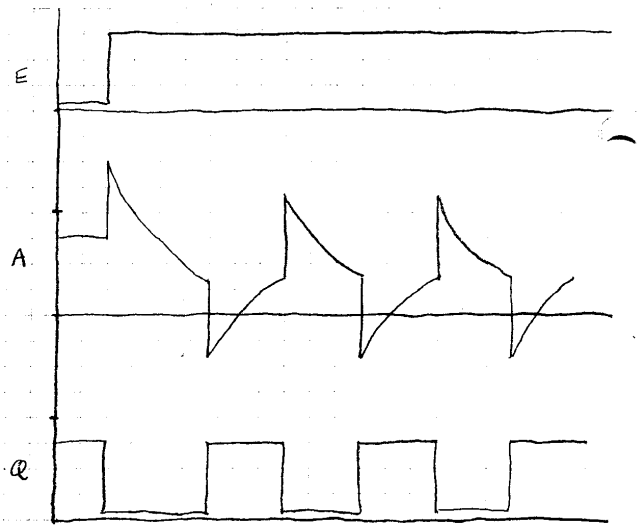
2) One-Shot



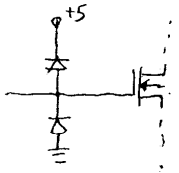


3) Gated Oscillator





Note R'~10R needed due to integrated protection diodes on inputs



This oscillator is more sensitive to power supply than the 7555. However it may be more convenient as an add-on to an already existing circuit if there are some unused gates.

25)

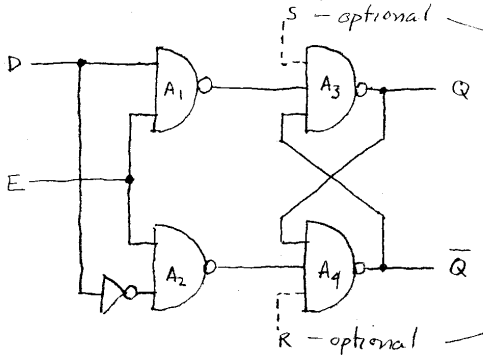
There are two problems with the simple RSFF that prevent its use in more complicated circuits

i) S & R need to change simultaneously

ii) Must avoid S = R = 1.

Two circuits which address these problems (– not completely)

1) D-type Transparent Latch

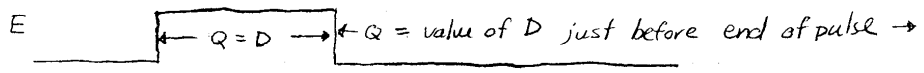


Function Table

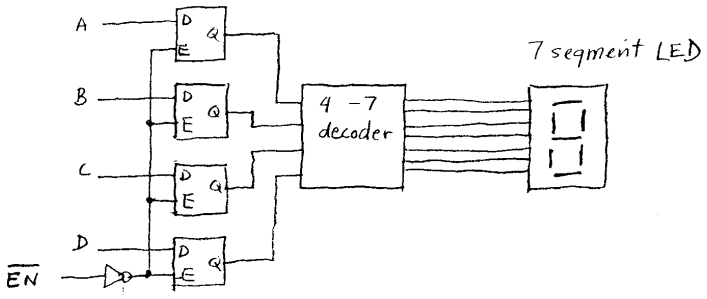
D	E	Q	$\overline{Q}$
x	0	no change from previous	
x	1	D	$\overline{D}$
0	⊓	0	1
1	⊓	1	0

The situation of asserting inputs (ie middle input=0) to both A_3 & A_4 is avoided by the inverter in front of A_2. The synchonization of inputs is accomplished by the enable E input.

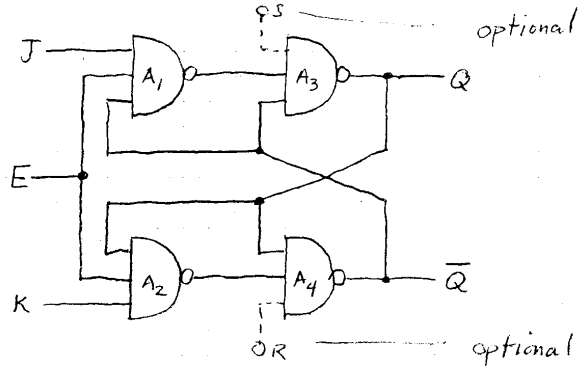
Note: the meaning of the lines in the Function Table where E = ⊓ is that if D has the given value just before the end of the pulse then Q assumes the given value just after the pulse.



This kind of Latch is found in the HP displays used in lab.



2) JK Latch – This turns out to have a serious flaw (26)



J	K	E	$Q^{(n+1)}$ (value after E pulse)
×	×	0	$Q^{(n)}$
0	1	⊓	0
1	0	⊓	1
* 1	1	⊓	$\bar{Q}^{(n)}$

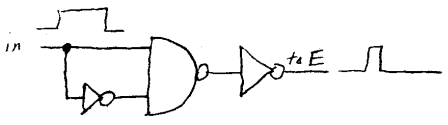
* indeterminate if E too long

Again a synchonization pulse E = ⊓ is used to insure that inputs to A_3 & A_4 change simultaneously. The asynchronous (optional) Set and Reset act independent of E and, in general, should only be used to define initial conditions. The function table can be verified from the truth table below which gives Q^{n+1} the output just after E pulse for the input cond^n $J^{(n)}$, $K^{(n)}$.... existing just before the end of the E pulse.

Truth Table for JK Latch

$J^{(n)}$	$K^{(n)}$	$Q^{(n)}$	$\bar{Q}^{(n)}$	$Q^{(n+1)}$	
0	0	0	1	$Q^{(n)}$	$Q^{(n)}$ "Hold"
0	0	1	0	$Q^{(n)}$	
1	0	0	1	1	1
1	0	1	0	$Q^{(n)}$	
0	1	0	1	$Q^{(n)}$	0
0	1	1	0	0	
1	1	0	1	1	$\bar{Q}^{(n)}$ "Toggle"
1	1	1	0	0	

Note a race-around condition occurs when J=k=E=1; the circuit will oscillate between $Q\bar{Q}$ = (01) and (10) with period $\sim 2\tau_{pd}$!! When E→0 it is indeterminate which state the circuit will end up in. A way to avoid this is to insure that the E pulse is very short, eg. use edge detector from pg 24

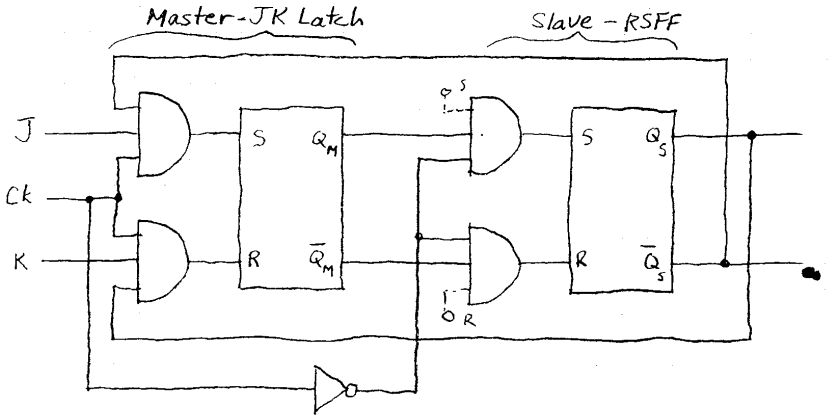


- only ↑ edge produces a pulse
- width ~ τ_{pd} of inverter.

(21)

Master - Slave JK Flip Flop

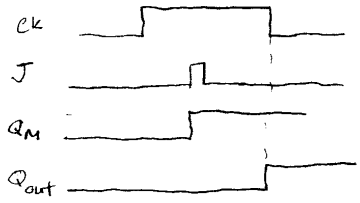
The oscillation problem in the Toggle state of the JK latch can be cured by breaking the feedback w/ the Master - Slave configuration.



When Ck=1 the master is enabled and it follows the JK latch Truth Table with $Q^{(n)}$ being provided by the slave outputs Q_S. When Ck=0 the master holds and the slave responds w/ Q_M as input according to the RSFF table on pg 22. Therefore the Master-Slave JKFF follows the function table at the top of pg 26. Note that the outputs change only on the Negative-going edge of the clock Ck.

However there is still a problem with this circuit! J and K must remain constant during Ck pulse otherwise the inputs before the pulse $\neq$> state after pulse according to table.

eg. suppose K=0 and the following seq.



Since J=K=0 before Ck pulse

Table => $Q^{(n+1)} = Q^{(n)}$.

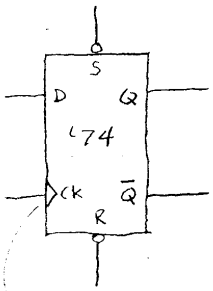
Instead $Q^{(n+1)} \to 1$, "one's catching."

(27)

Edge-Triggered Flip Flops

Generally these are the most convenient to apply, compared to the level sensitive types (D latch JK latch) and M/S configurations, because the output changes *after* a clock edge according to the input(s) present just *before* the clock edge. Two major types used in our Lab.

1) Positive-Edge Triggered D-type Flip Flop



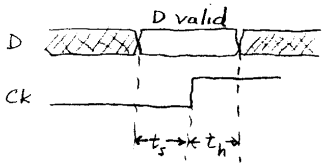
> means edge-triggered

S	$\bar{R}$	Ck	D	$Q^{(n+1)}$	$\bar{Q}^{(n+1)}$
0	1	×	×	1	0
1	0	×	×	0	1
1	1	↑	0	0	1
1	1	↑	1	1	0
1	1	↓	×	$Q^{(n)}$	$\bar{Q}^{(n)}$

Note: i) Output changes only on the positive sloping edge of Ck.

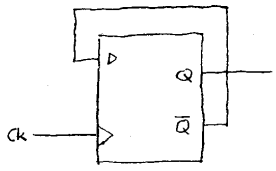
ii) S + R overide Ck, they act shortly after applied independent of Ck. (This is called asynchronous) the inversion symbol ○ means the specified action takes place when signal is *low* ie S = 0 to set Q = 1.

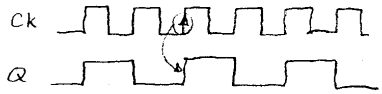
iii) Set up and Hold times must be satisfied.



(30)

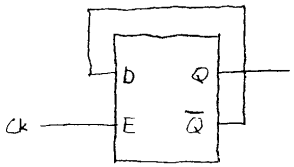
Consider T-Type (Toggle) Flip Flop built from DFF

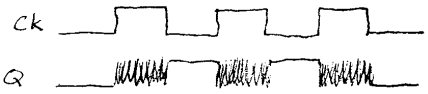




Divides Ck frequency by 2

Suppose one tried to use a D-Latch instead

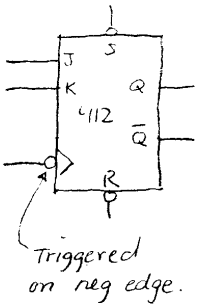




When E=1 circuit oscillates

⇒ unpredictable state when E→0.

2) Negative Edge Triggered J K Flip Flop



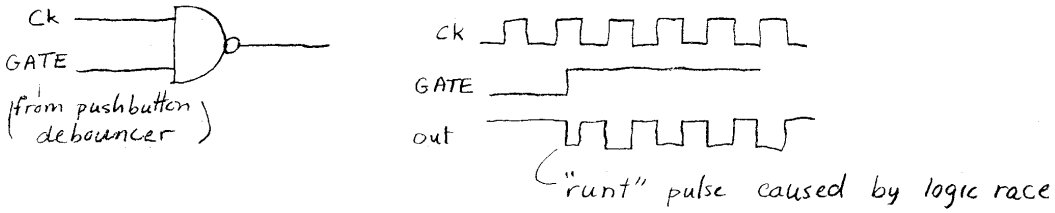
S	R	J	K	Ck	$Q^{(n+1)}$	
0	1	×	×	×	0	
1	0	×	×	×	1	
1	1	0	1	↓	0	
1	1	1	0	↓	1	
1	1	0	0	↓	$Q^{(n)}$	"Hold"
1	1	1	1	↓	$\bar{Q}^{(n)}$	"Toggle"

(31)

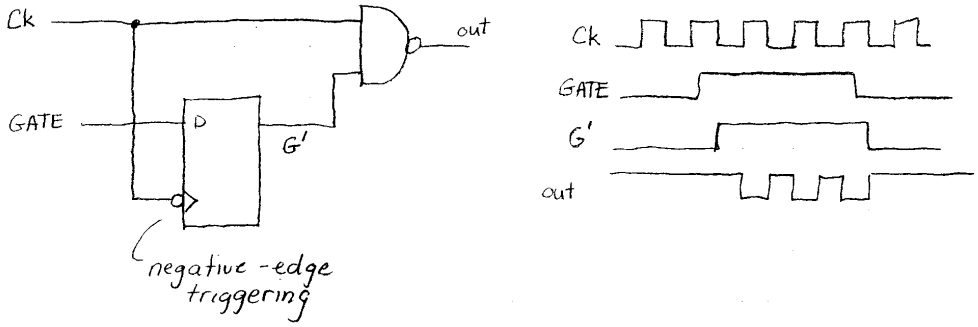
Edge-Triggered Flip Flop Applications

1) Synchronizer: in a clocked sytem one often wants to start the action of a circuit w/ an asynchronous signal e.g. a push button which starts the clock

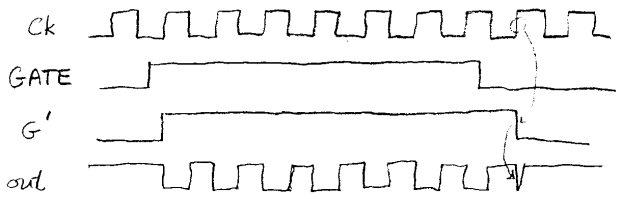
First, an example of a bad synchronizer



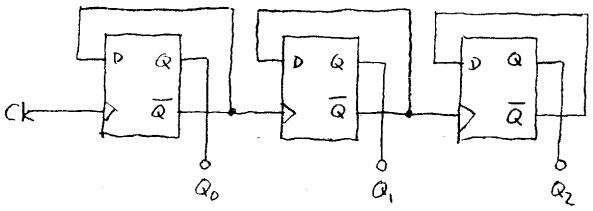
A better synchronizer would have GATE changing only while Ck = 0 since then the output is definitely High.

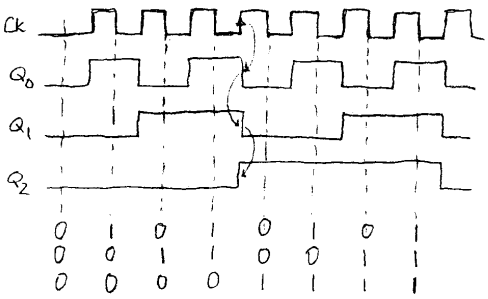


Note if one instead used a positive-edge triggered FF there would always be a glitch at the end of GATE because G' goes low slightly after Ck ↓



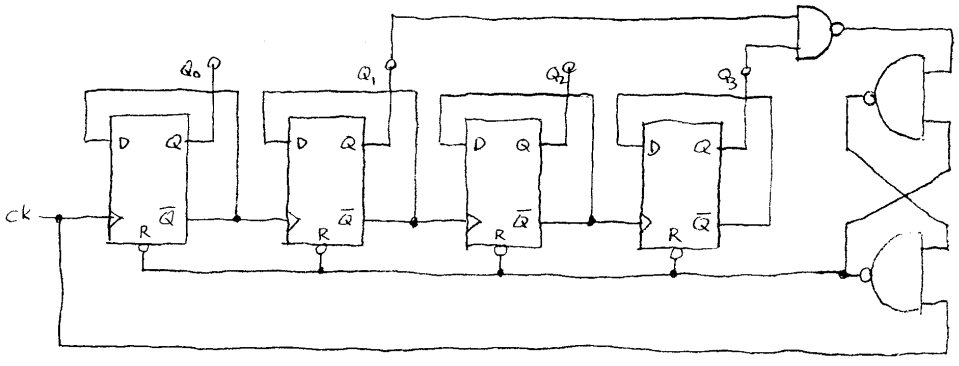
2) Ripple Counter – counts Ck pulses using T-type FF (32)





Note it takes ~$3\tau_{pd}$ for bits to settle to final values. This would be OK for a frequency divider. However in some applications, e.g. addressing memory, one desires all bits to change at nearly the same instant. – synchronous counter discussed later.

To make a divide-by-N add a feedback gate so that at count N all FF are asynchronously reset. Since the propagation delay from Clear to Q may vary from FF to FF the reset pulse should be stored to insure that all FF are reset before the reset signal ends. As an example consider a divide-by-10. Will need 4 FF (max count=16)



(33)

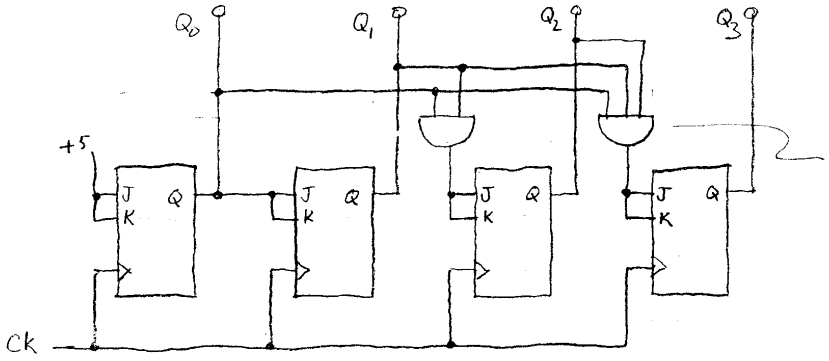
Synchronous Counters

In many cases one would like the bits of a counter to all change simultaneously in response to Ck rather than change sequentially as happens in a ripple counter. To do this one needs to "cook-up" logic that has the present state of the counter bits $\{Q_n\} = Q_0 Q_1 \cdots Q_n$ as inputs and produces new $\{D_n\}$ or $\{J_n K_n\}$ for the FF in time for the next Ck.
For example consider the sequence of states in a 4 bit (natural) binary counter

```
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
  ⋮
  ⋮
1 1 0 1
1 1 1 0
1 1 1 1
```

A rule governing this sequence is Q_m toggles only when $Q_i = 1$ for all $i < m$.

It turns out to easiest to realize this w/ JF Flip Flops since they toggle or hold based on JK inputs.



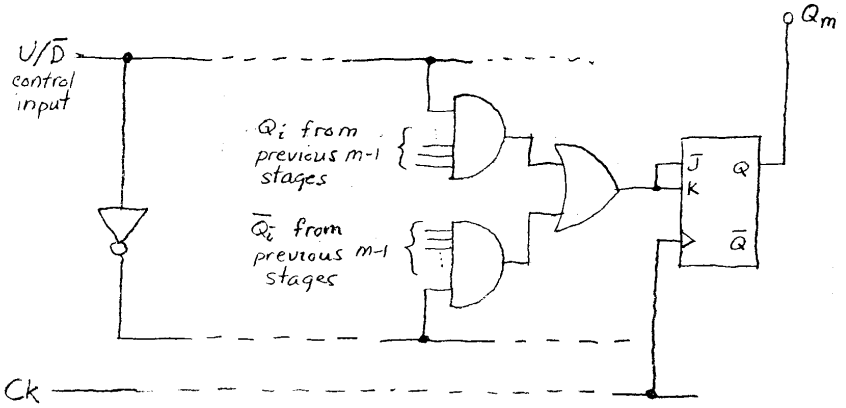
Note that ideally all Q_i change simultaneously when Ck $\downarrow$.
Therefore a divide-by-n made with a synchronous counter will operate at much higher frequency than w/ ripple counter.

(34)

Other features available in IC counters

- "Direction" Up/Down Control

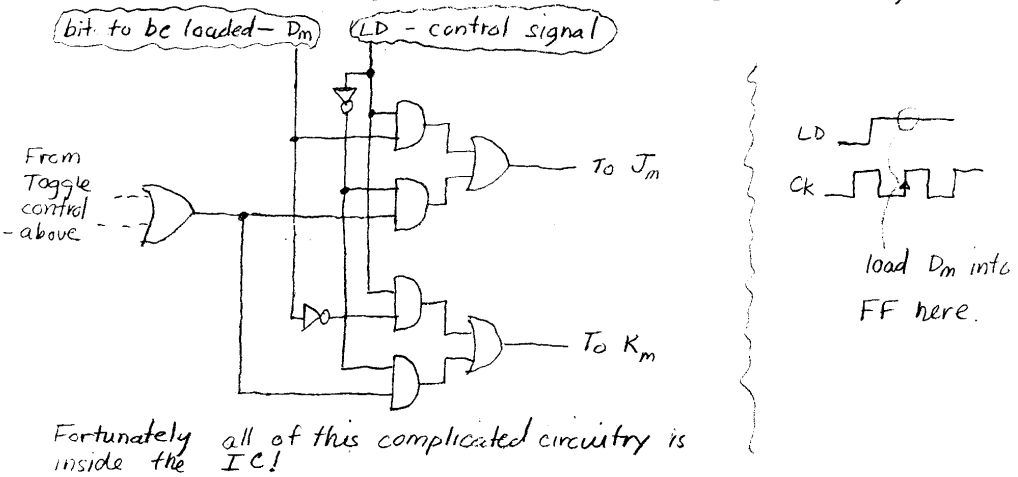
  From the counting sequence one can see that the rule for counting down is Q_m toggles when all previous $Q_i = 0$. So the logic for driving the JK inputs for the m^{th} FF would look like



  U/D̄ must be asserted some "setup time" t_s before Ck↑ in order to work properly.

- Parallel Load

  Allows you to preset the counter to a particular value. Can be synchronous ie takes effect when Ck↑ or asynchronous ie takes effect immediately. An example of synchronous loading the JK inputs might look like;

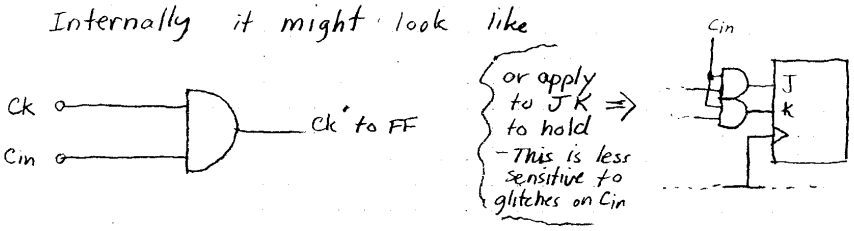


  Fortunately all of this complicated circuitry is inside the IC!

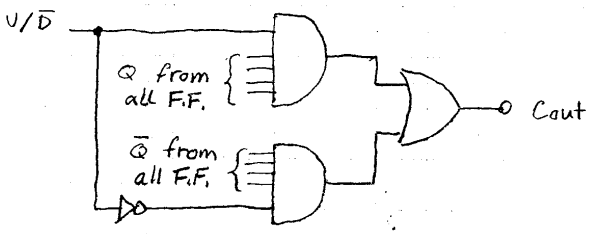
IC Counter Features c'td (35)

- Carry-in (Cin) and Carry-out (Cout)
  These are primarily used for connecting two IC n-bit counters to make a 2n-bit counter.

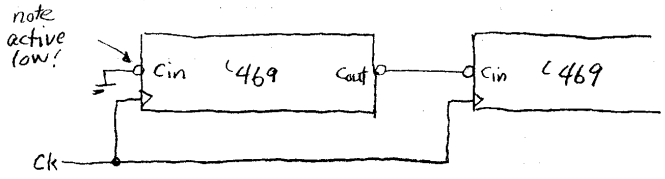
  i) If Cin is False then counting should be inhibited.
  Internally it might look like



  ii) Cout is an output which is true when the terminal count of the counter is reacheed, ie 0 if counting down.



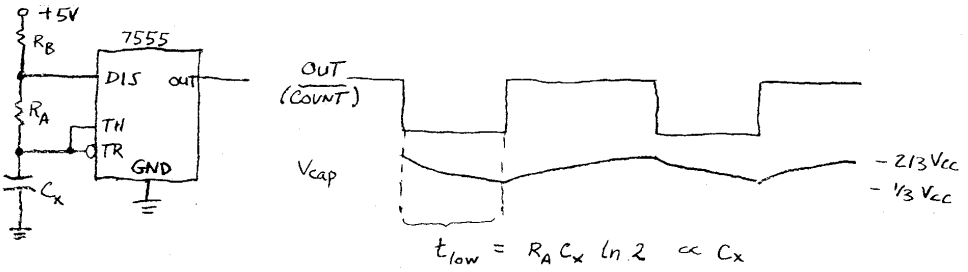
Making a 2n-bit counter out of two n-bit counters is usually this simple



(56)

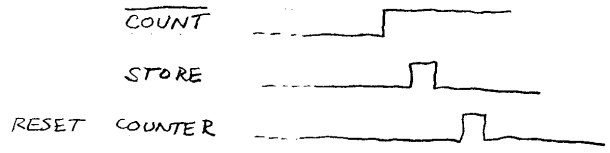
Counters are useful in measurement if the physical quantity to be measured can be "converted" into a pulse whose duration is ∝ to the value of the quantity. A counter then counts the number of Clock "ticks" (Ck↑) during the pulse.

The example built in Lab is a capacitance meter



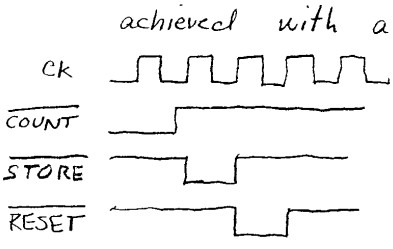
Count Clocks while OUT = LOW ⇒ C_x measurement,

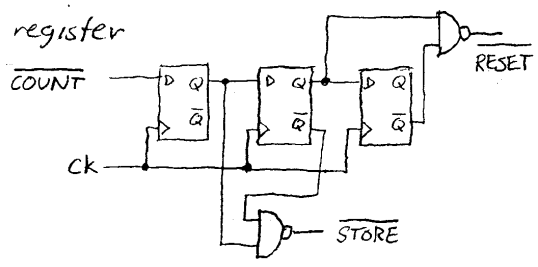
The main detail that one needs to take care of is how to store the count at the end of one measurement cycle reset the counter for the next cycle. The desired pulse sequence looks like



Note if the storage register is edge sensitive STORE and RESET could occur at the same time if the required hold time of the register's FF = 0. If the register is level sensitive you must have RESET occuring later to avoid STORE'ing zero!

In the Lab the necessary delay and pulse formation is achieved with a shift register



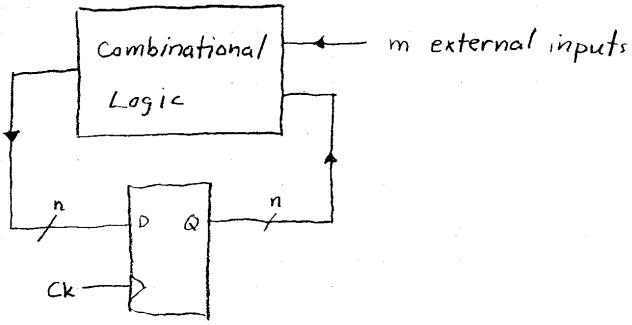


(31)

State Machines

A collection of n flip-flops (n-bit register) can have 2^n possible states ie values of $Q_0, Q_1, Q_2 \ldots Q_n$. A state machine is a clocked digital sytem whose $\{Q_i\}$ move through the state space. The trajectory may be determined by the $\{Q_i\}$, external inputs or both. In the former, more restrictive, case the trajectory is closed ie it repeats. This is a generalized counter. The latter case is somewhat like a computer.

The general architecture of a state machine is

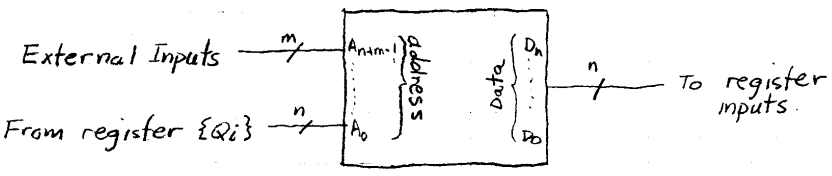


n-bit register

Note i) It is important that the register be edge triggered so that $\{Q_i\}$ remain constant while all signals propagate through the combinational logic.

ii) The combinational logic could be replaced by either n (n+m)-line multiplexers (see pg 19) or a 2^{n+m} (address) × n-bit memory. The latter is more general but is usually somewhat slower than the decoders or using combinational logic.

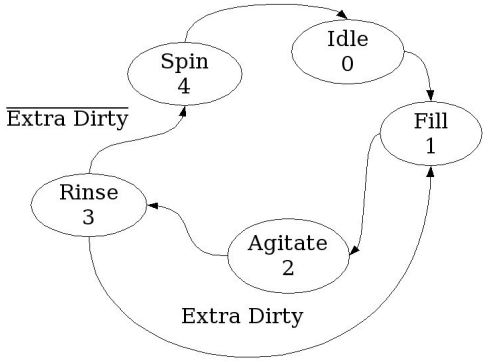
Use of Memory



8.24, 25 AE-10, 9.3, 4

We can think of a state machine as somewhat like counter with 2^n possible values, but state machine can progress in a more general way (not limitied to monotonic progressions).

Example: washing machine



First detailed example: 2-bit gray code counter

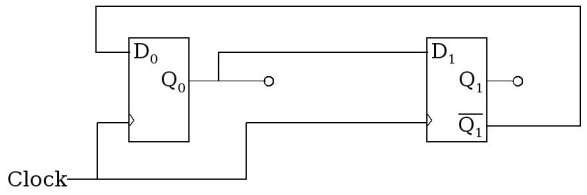
Count progression is: 00, 01, 11, 10, 00, ... These represent out states and their relative ordering.

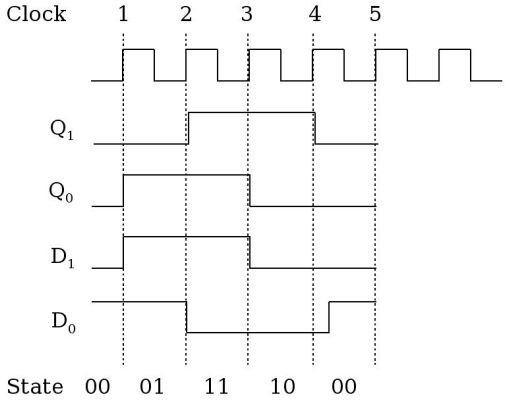
To construct this device: write a truth table for input/outputs: inputs=current stare info, outputs = next state info.

	Previous Q_i input for comb^n logic		Next Q_i output of comb^n logic		
State	Q1	Q0	D1	D0	
0	0	0	0	1	*These are the inputs we need to get to the next state*
1	0	1	1	1	
2	1	1	1	0	
3	1	0	0	0	

Devise necessary logic from the T Table to generate D's from Q's:

$$D_0 = \overline{Q_1}\,\overline{Q_0} + \overline{Q_1}\,Q_0 = \overline{Q_1}\cdot(\overline{Q_0}+Q_0) = \overline{Q_1}$$
$$D_1 = \overline{Q_1}\,Q_0 + Q_1\,Q_0 = Q_0\cdot(\overline{Q_1}+Q_1) = Q_0$$





An example of a State Machine as a Generalized counter is a Grey Code counter. Suppose we want to have 8 states ie count 0–7. => Need 3 bit register. (38)

Write a Truth Table for the register D inputs and Q outputs

Previous $\{Q_i\}$ (input to Comb^{a} logic)			Next $\{D_i\}$ (output of Comb^{a} logic)		
Q_2	Q_1	Q_0	D_0	D_1	D_2
0	0	0	1	0	0
0	0	1	1	1	0
0	1	1	0	1	0
0	1	0	0	1	1
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	0	0	1
1	0	0	0	0	0

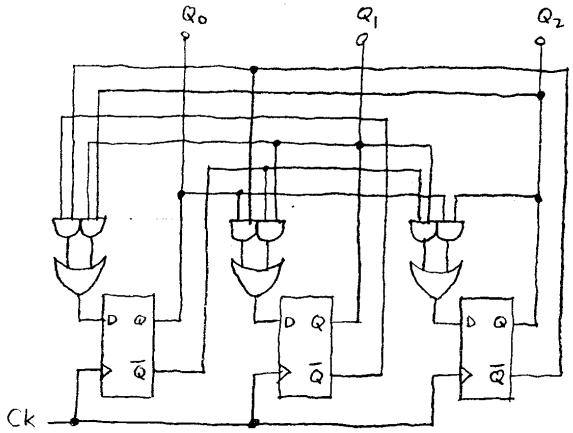
Now simply read-off minterms from table

$$D_0 = \bar{Q}_0 \cdot \bar{Q}_1 \cdot \bar{Q}_2 + Q_0 \cdot \bar{Q}_1 \cdot \bar{Q}_2 + \bar{Q}_0 \cdot Q_1 \cdot Q_2 + Q_0 \cdot Q_1 \cdot Q_2$$

$$= \bar{Q}_1 \cdot \bar{Q}_2 + Q_1 \cdot Q_2$$

$$D_1 = Q_0 \cdot \bar{Q}_2 + \bar{Q}_0 \cdot Q_1$$

$$D_2 = \bar{Q}_0 \cdot Q_1 + Q_0 \cdot Q_2$$



Note The trajectory of this state machine fills the entire space. If instead one made a divide-by-k, where $k < 2^n$ then some states would never be reached under normal operation – called excluded states.

One must take care, that if an excluded state is reached (21)
(e.g. at power-up) then the comb^{n} logic will insure that eventually a state within the trajectory will be reached.

For example consider a down-counter following the sequence $3 \to 2 \to 1 \to 3 \ldots$ ie $(11) \to (10) \to (01) \to (11)$. Excluded state is (00).
The truth table is

Previous $\{Q_i\}$		Next $\{D_i\}$	
Q_1	Q_0	D_1	D_0
1	1	1	0
1	0	0	1
0	1	1	1

$\Rightarrow$

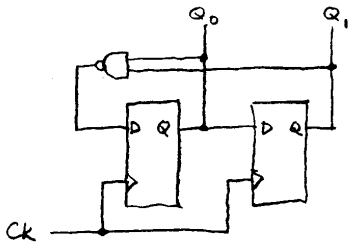
$$D_0 = Q_0 \cdot \overline{Q_1} + \overline{Q_0} \cdot Q_1 = Q_0 \oplus Q_1$$
$$D_1 = Q_0 \cdot Q_1 + Q_0 \cdot \overline{Q_1} = Q_0$$

Note that if one uses $D_0 = Q_0 \oplus Q_1$ then the excluded state $(00) \to (00)$!!
ie if the circuit ever winds up in (00) it will stay there.
A way to avoid this problem is to add minterms corresponding to excluded states

i.e. let $D_0 = Q_0 \cdot \overline{Q_1} + \overline{Q_0} \cdot Q_1 + \overline{Q_0} \cdot \overline{Q_1} + \overbrace{\overline{Q_0} \cdot \overline{Q_1}}^{=0 \text{ for states in trajectory}}$

$$= (Q_0 + \overline{Q_0}) \cdot \overline{Q_1} + (Q_1 + \overline{Q_1}) \cdot \overline{Q_0} = \overline{Q_0} + \overline{Q_1} = \overline{Q_0 \cdot Q_1}$$

So a circuit for the down-counter which doesn't get locked in (00) is

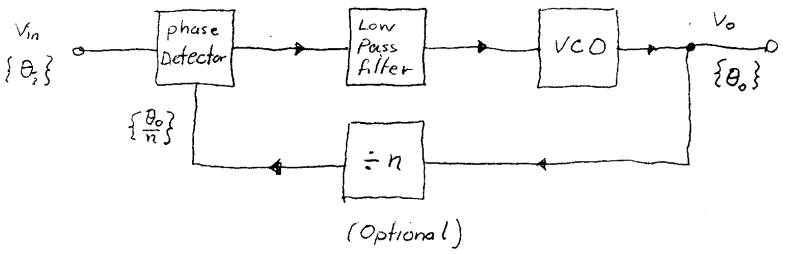


Phase Locked Loop

(1)

When we analyzed OPAMP feedback circuits recall that the OPAMP compared an input voltage v_{in} to a feed back modified version of its output voltage, βv_o. The operation was such as to reduce the error $v_{in} - \beta v_o$ to zero.

The Phase Locked Loop is also a feedback circuit but the quantity which is being compared is the phase of oscillatory signals. The basic topology looks like the following diagram.



V_i and V_o are oscillating signals

$V_i \sim \sin(\omega_c t + \theta_i(t))$

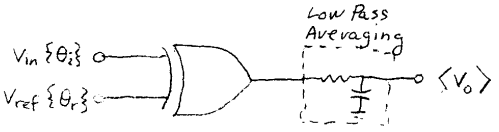
$V_o \sim \sin(\omega_c t + \theta_o(t))$

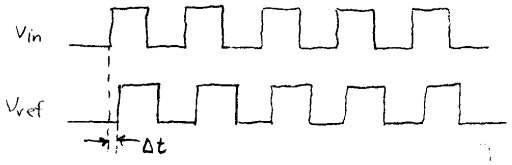
↑ common "carrier" frequency. If different freqs, then $\theta_i - \theta_o \sim \pm \Delta\omega$.

The output of the phase detector is a voltage related to the difference in phases $\theta_i - \frac{\theta_o}{n}$. After filtering this voltage is applied to voltage controlled oscillator VCO to modify θ_o to give a constant phase difference $\theta_e = \theta_i - \frac{\theta_o}{n}$ at the phase detector.

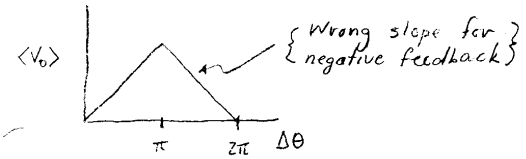
Type I Phase Detector

1) XOR - Used w/ 50% duty factor square waves





$\Delta\theta \equiv \theta_i - \theta_r = \omega_c \Delta t$



$$\langle V_o \rangle = \begin{cases} V_{cc} \frac{\Delta\theta}{\pi} & 0 \le \Delta\theta \le \pi \\ V_{cc}\left(2 - \frac{\Delta\theta}{\pi}\right) & \pi \le \Delta\theta \le 2\pi \end{cases} \quad (1)$$

(2)

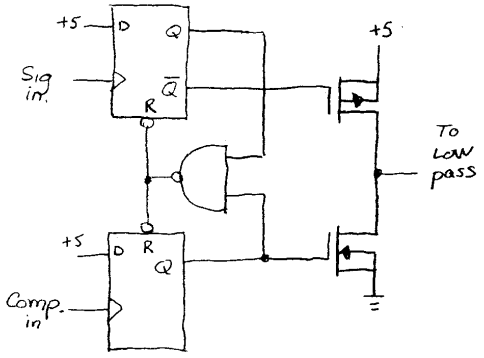
Note: i) if ref is at a harmonic or subharmonic of V_{in} then $\langle V_o \rangle$ is unchanged. Therefore PLL can lock onto harmonics of input frequency when using Type I detectors

ii) Initially assume VCO is free running before input applied. Then $\langle V_o \rangle = \frac{1}{2} V_{cc}$ $\Rightarrow$ VCO will go to center frequency.

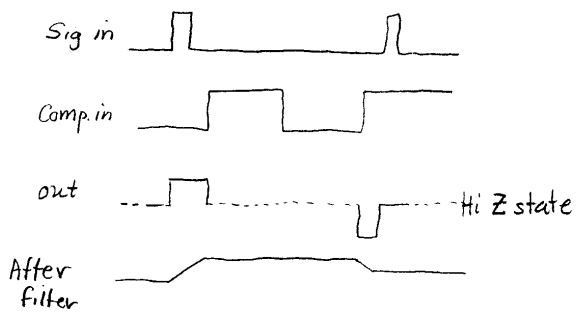
iii) When loop is in lock the $\Delta\theta$ corresponds to that needed to give the necessary VCO input voltage for the input frequency, ie if input freq near max of VCO's range then $\Delta\theta \to \pi$.

Type II Phase Detector

Edge sensitive network - based on Flip-Flops



Assume initially Sig.in led Comp.in,



Advantages: Correction pulses generated only when needed.
Doesn't lock on harmonics
$\Delta\theta = 0$ when locked - indep of f

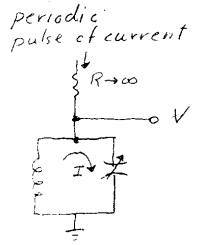
Disadvantage: sensitive to noise and distortion.

wiggle interpreted as two edges.

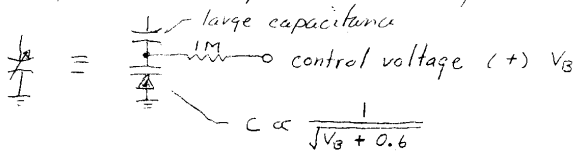
VCO and FM Generation

Claim: suppose control voltage (ie audio) = $f(t)$. Then output waveform $\sim \cos(\omega_c t + \int f(t)dt)$. Two examples;

a) Sinusoidal Oscillators – usually have an LC tank circuit in feedback loop.



The variable capacitance is most easily realized with a varactor – back biased diode whose junction capacitance depends on voltage.



Considering the LC tank circuit above; $\dot{I} = C\ddot{V} = -\frac{1}{L}V$ or

$$LC_0\left(1 + \frac{\Delta C}{C_0}\right)\ddot{V} = V \qquad (1)$$

where ΔC = change in capacitance due to modulating (audio) signal $\Delta V_B = f(t)$.

Assume a solution of the form; $V \sim e^{j\Phi(t)}$ where Φ is assumed to have primarily a linear dependence. In particular approximate $\frac{d^n}{dt^n}\Phi = 0$ $n > 1$. (i.e. assume there are only smooth changes in $\dot{\Phi}$)

Putting this solution into (1)

$$\Rightarrow \dot{\Phi} = \frac{1}{\sqrt{LC_0\left(1+\frac{\Delta C}{C_0}\right)}} \simeq \frac{1}{\sqrt{LC_0}}\left(1 - \frac{1}{2}\frac{\Delta C}{C_0}\right) = \omega_c + K_0 f(t) \qquad (2)$$

where $\omega_c = \frac{1}{\sqrt{LC_0}}$ and $K_0 = \frac{1}{4}\frac{\omega_c}{(V_B + 0.6)}$.

($\frac{\omega_c}{2\pi}$ called the carrier frequency and K_0 is the VCO "gain" in $\frac{\text{rad/sec}}{\text{Volt}}$)

$$\therefore \quad V \sim \mathrm{Re}\, e^{j\Phi(t)} = \cos(\omega_c t + K_0 \int f(t)) \qquad (3)$$

($K_0 \int f(t)$ is called the frequency deviation)

Compare (3) to text sec 13.18.

(4)

b) Square Wave Oscillators - Here voltage controlled current sources are used to charge a capacitor.

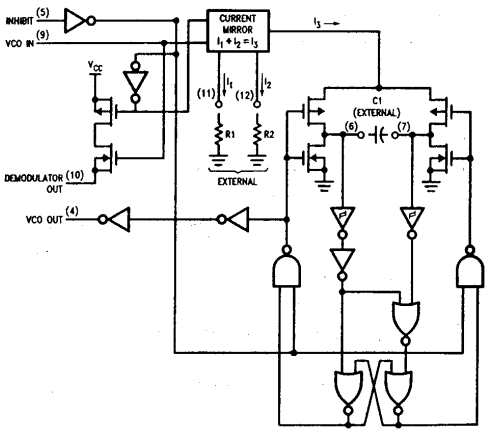
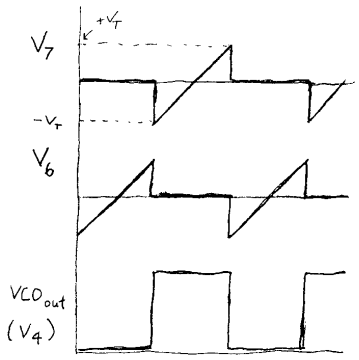


FIGURE 2. Logic Diagram for VCO



The capacitor voltages V_6 & V_7 are linearly related to the phase Φ,

e.g. $\Phi = \begin{cases} \frac{\pi}{2}\frac{V_7}{V_T} & \text{while } V_6 = 0 \\ \frac{\pi}{2}\left(2+\frac{V_6}{V_T}\right) & \text{while } V_7 = 0 \end{cases}$

In either case $\dot{\Phi} = \frac{\pi}{2}\frac{\dot{V}_{6,7}}{V_T} = \frac{\pi}{2V_T}\frac{I_3}{C_1}$.

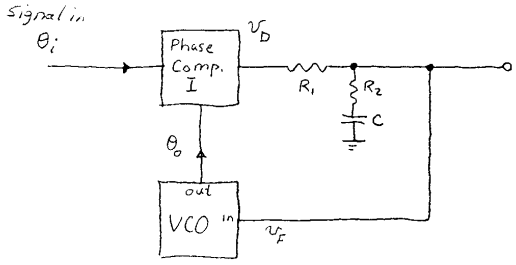
But I_3 is set by a current mirror; $I_3 = I_3^{D.C.} + g_m f(t)$ $(f(t) = VCO\ input)$.

$\Rightarrow \dot{\Phi} = \omega_c + K_0 f(t)$, where $\omega_c = \frac{\pi}{2}\frac{I_3^{D.C.}}{V_T C_1}$ and $K_0 = \frac{\pi}{2}\frac{1}{V_T}\frac{g_m}{C_1}$.

$\therefore VCO_{out} \sim H[\cos(\omega_c t + K_0 \int f(t))]$ (4)

Here H is the Heavyside function – turns sine wave into square wave

Closing the Loop - Frequency Demodulation (5)



Assume signal input $V_i(t) \sim H[\sin(\omega_c t + \theta_i(t))]$

and VCO output $VCO_{out}(t) \sim H[\cos(\omega_c t + \theta_o(t))]$

Note a 90° phase shift between input and VCO output is included explicitly since that phase difference corresponds to the mid value of the Type I phase comparator output.

Phase Comp output : $v_D = K_D(\theta_i - \theta_o)$

Filter output : $v_F = K_F v_D$ where $K_F = \frac{1+j\omega\tau_2}{1+j\omega(\tau_1+\tau_2)}$, $\tau_n = R_n C$.

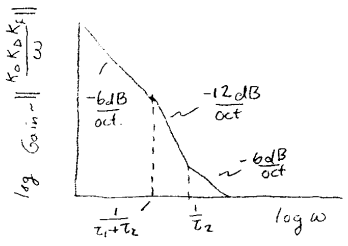
VCO output : $\dot{\theta}_o = K_o v_F \Rightarrow \theta_o = \frac{1}{j\omega} K_o v_F$

(Note the ω in above expressions relates to the time dependence of θ_i & θ_o)

The loop equation is $\theta_o = \frac{1}{j\omega} K_o K_F K_D (\theta_i - \theta_o)$,

$$\Rightarrow \quad \theta_o = \frac{\frac{1}{j\omega} K_o K_D K_F}{1 + \frac{1}{j\omega} K_o K_D K_F}\, \theta_i \qquad (5)$$

The resistors R_1 & R_2 are usually chosen $R_2 << R_1$ to give a Bode plot



which has a slope ~ -6dB/oct when Gain passes through 0dB. Just like OPAMP case,

Using the expression for K_F, equation (5) becomes (6)

$$\frac{\theta_o}{\theta_i} = \frac{K_o K_D (1 + j\omega\tau_2)}{-\omega^2(\tau_1+\tau_2) + j\omega(1+\tau_2 K_o K_D) + K_o K_D}$$

$$= \frac{1 + j\omega\tau_2}{-\frac{\omega^2}{\omega_n^2} + 2j\delta\frac{\omega}{\omega_n} + 1} \qquad (6)$$

where $\omega_n = [K_o K_D/(\tau_1+\tau_2)]^{1/2}$ (loop Natural Frequency)

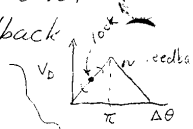
and $\delta = \frac{1}{2}\sqrt{\frac{1}{(\tau_1+\tau_2)K_o K_D}}(1+\tau_2 K_o K_D)$ (Damping Factor)

{ note $\delta \simeq \frac{1}{2}\omega_n\tau_2$ if $\tau_2 K_o K_D >> 1$ }

$$\simeq \frac{1 + 2j\delta\frac{\omega}{\omega_n}}{-(\frac{\omega}{\omega_n})^2 + 2j\delta\frac{\omega}{\omega_n} + 1}$$

Since the PLL is to be used to demodulate FM the loop bandwidth $\sim\omega_n$ should be made as large as possible in order to follow the signal. On the other hand it must not be excessively large to avoid noise. Since we're interested in audio we'll anticipate $\omega_n \simeq 2\pi \times 20kHz$.

Also in order for the loop to remain locked w/the input the phase error should be limited to $\lesssim 90°$. (Otherwise may aquire positive feedback



Consider sinusoidal FM;

$$\theta_i(t) = \Delta\omega \int e^{j\omega_m t}\,dt = \frac{\Delta\omega}{j\omega_m} e^{j\omega_m t}$$, where ω_m = modulating frequency
$\Delta\omega$ = amplitude of freq. deviation.

$$\Rightarrow \tilde{\theta}_i(\omega) = \frac{\Delta\omega}{j\omega_m}\delta(\omega-\omega_m) \qquad \text{(note: work w/ Fourrier Transforms)} \qquad (7)$$

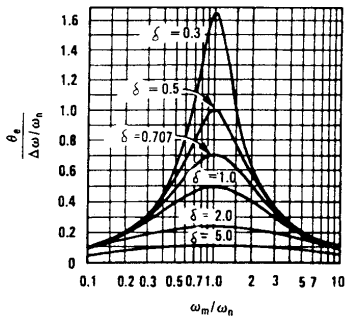
Using (6) and (7) the phase error can be approximated

$$\theta_e = \theta_i - \theta_o = (1 - \frac{\theta_o}{\theta_i})\theta_i \simeq \frac{-(\frac{\omega_m}{\omega_n})^2 \frac{\Delta\omega}{j\omega_m}}{-(\frac{\omega_m}{\omega_n})^2 + 2j\delta\frac{\omega_m}{\omega_n} + 1}$$ or

$$\frac{\theta_e}{\Delta\omega/\omega_n} = \frac{j\frac{\omega_m}{\omega_n}}{1 - (\frac{\omega_m}{\omega_n})^2 + 2j\delta\frac{\omega_m}{\omega_n}} \qquad (8)$$

(7)

A plot of eqn (8) is shown here →



Max. phase error θ_e occurs when a frequency of the modulating signal $\omega_m = \omega_n$ (Natural Freq. of loop).

At this frequency we desire

$$\|\theta_e\| = \frac{1}{2\delta}\frac{\Delta\omega}{\omega_n} \leq \frac{\pi}{2} \quad (\text{at } \omega_m = \omega_n) \qquad (9)$$

Eqns (6) and (9) can be used to set the filter component values.

Ex Phase Detector: $K_D = \frac{V_{cc}}{\pi} = 1.59$ V/rad for $V_{cc} = 5V$

VCO Data:

VCO_{in}	f
1.25V	33kHz
2.5V	98 kHz
3.75V	150 kHz

$$\Rightarrow K_O = \frac{2\pi(117\,kHz)}{2.5V} \simeq 3\times10^5 \frac{rad/sec}{V}$$

Choose $\omega_n \simeq 2\pi \times 10$kHz for audio, then from eqn (6)

$$\tau_1 + \tau_2 = \frac{K_O K_D}{\omega_n^2} = 1.2\times10^{-4}\ sec$$

A damping factor of $0.5 < \delta < 1$ usually gives good performance i.e. not too much ringing. Usually $\delta = 0.7$ is considered optimum. Also from eqn (6)

$$\tau_2 \simeq \frac{2\delta}{\omega_n} = 2.2\times10^{-5}\ sec$$

An acceptable set of values would be

$R_1 = 9.1k\Omega$ $C_1 = 10$ nf
$R_2 = 2.2k\Omega$

Since $\delta = .7$ the frequency deviation should be limited to (using eqn (9))

$$\Delta\omega \leq \pi\delta\omega_n = 1.2\times10^5 \text{ rad/sec} \quad \text{or} \quad \Delta f \simeq 20\,kHz$$

For $f_c = 100$kHz (carrier) this represents a 20% modulation.

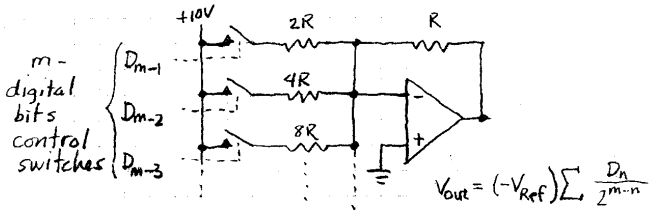
Data Conversion

The measurement of some physical property by an electronic transducer is usually an analog voltage or current which is related (1-to-1) to the value. Subsequently one wishes to either store the measured value or use it in a further computation. In either case it turns out to be most convenient to convert the measurement to a number - binary or otherwise.

Digital to Analog Conversion (DAC)

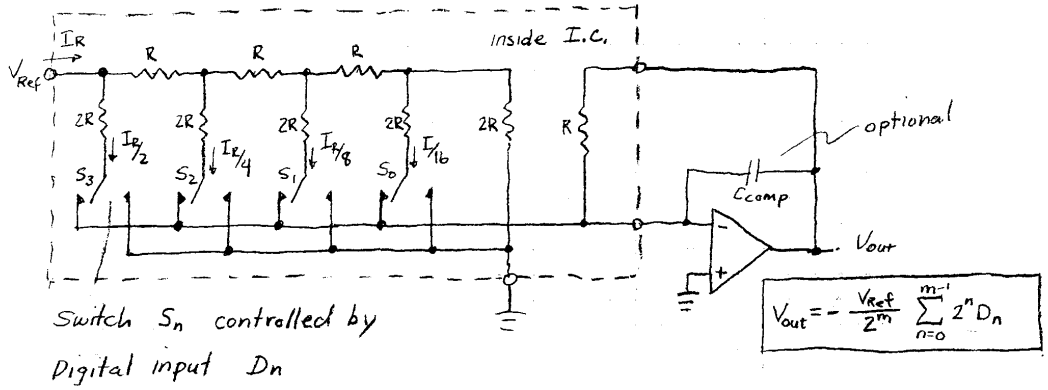
Begin first with the inverse operation: given a number convert it to a voltage or current. (This forms the basis of all feedback Analog-to-Digital converters) It turns out, due to the KCL that digital conversion to current is the most natural.

A first attempt might be the summing-amp approach from Ex 4.5



The main disadvantage of this scheme is that an n-bit DAC would require n+1 different resistor values, each one having an absolute accuracy $\simeq \frac{1}{2}$ smallest value, ie for a 10-bit converter the largest resistor would need 0.05% accuracy!

A more elegant scheme which uses only two resistor values is the R-2R ladder. A 4-bit version is shown below

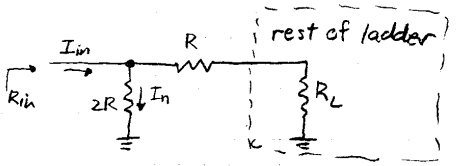


Switch S_n controlled by Digital input D_n

(2)

How the R-2R ladder works.....

At each "rung" of ladder the circuit looks like



If $R_L = R$ then

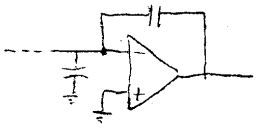
i) from l.h.s. $R_{in} = R$

ii) $I_n = \frac{1}{2} I_{in}$

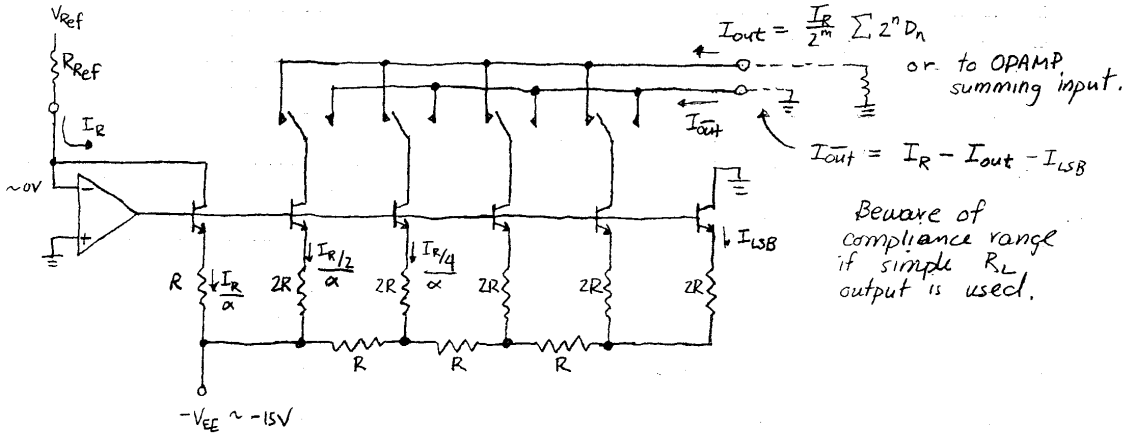
Note i) Uses CMOS analog switches

ii) Both summing lines must be held at ground (real or virtual) to work properly as a DAC.

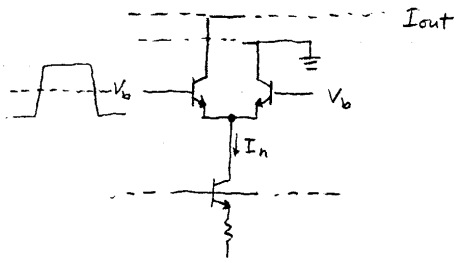
iii) Summing line usually has a large capacitance to ground which can be compensated-for somewhat by C_{comp}.



One can avoid the external OPAMP, which satisfies ii above, by using a current-source DAC.



Switch could be implemented by long-tailed pair



2a

DAC Error Specifications

Suppose a voltage output although same results w/ obvious modifications will apply to current out. Ideally the voltage levels out should be related to digital code by

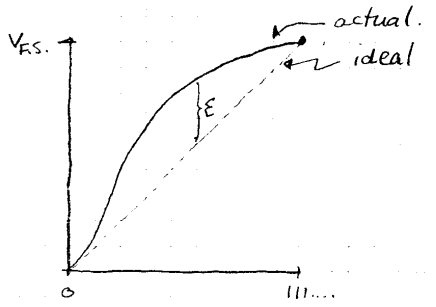
$$V_{out} = V_{REF} \times \frac{1}{2^m} \sum_{n=0}^{m-1} D_n \cdot 2^n .$$

Between adjacent numbers V_{out} should change by an amount corresponding to the least-significant bit, ie $\Delta V_{out} = \frac{V_{REF}}{2^m} \equiv \underline{V_{LSB}}$.

Also note that min $V_{out} = 0$ and max $V_{out} = V_{REF} - V_{LSB} \equiv \underline{V_{F.S.}}$ (full scale)

(ie max $V_{out} / V_{REF} = \frac{1}{2^m} \sum_{0}^{m-1} 2^n = \frac{1}{2^m}\left(\frac{1-2^m}{1-2}\right) = 1 - \frac{1}{2^m}$)

Consider transfer curve V_{out} vs digital code



1) Scale error $\equiv$ max $V_{out} \neq V_{F.S.}$ defined above
2) Integral Nonlinearity $\equiv$ max ε / V_{LSB} expressed in units of LSB's
3) Differential Nonlinearity $\equiv$ max $|V(n) - V(n-1)| / V_{LSB}$ units of LSB's
4) Monotonicity

Comparisons 26

i) The current mode DAC is unipolar ie $I_R > 0$ required, while analog switch DAC is bipolar ie $V_{ref} \lesseqgtr 0$ ($I_R \lesseqgtr 0$) allowed.

ii) Although both DACs require only two resistor values the current mode DAC must have m different sized transistors so that I_S (from Ebers Moll) scales. This is needed to insure all ladder transistors have same V_{BE}, since $\Delta V_{BE} \sim 60mV$ for $\frac{I_1}{I_2} = 10$.

iii) Current-mode DAC output capacitance ~ constant while analog-switch DAC capacitance varies w/ code. Only affects high speed operation.

iv) As presented both DAC are "multiplying" ie $V_{out} \propto V_{ref}$. Some analog-switch DACs have V_{REF} and output amplifier integrated.

Analog to Digital Conversion

Given a voltage range defined by limits V_{RT} & V_{RB} where $V_{RT} > V_{RB}$. Divide the range into 2^m equal intervals having boundaries

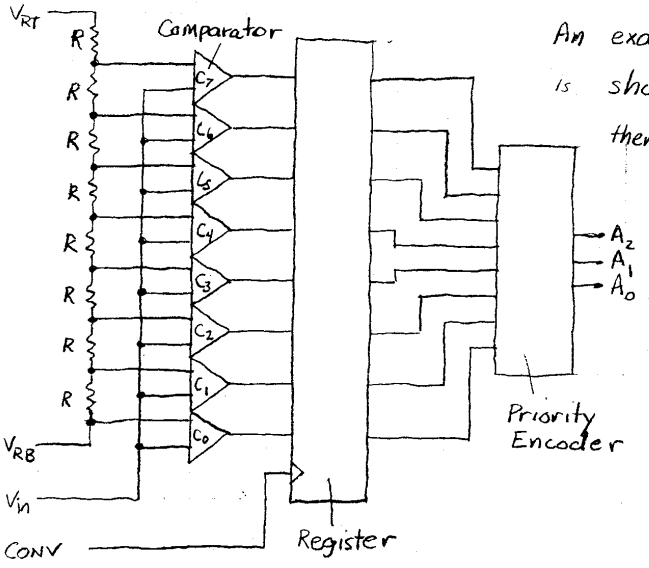
$$V_k = V_{RB} + (V_{RT} - V_{RB})\frac{k}{2^m}, \quad k = 0, 1, \ldots 2^m - 1.$$

The task of the ADC is to find k such that $V_k < V_{in} < V_{k+1}$

There are two broad categories; sampling (converts $V_{in}(t_{meas.})$) and integrating (converts $\langle V_{in} \rangle$). Examples of each are given below

1. Parallel Encoding ("Flash" ADC)

Conceptually simple but requires lots of circuitry e.g. 2^m comparators!



An example of a 3-bit Flash ADC is shown to left. If $V_k < V_{in} < V_{k+1}$ then C_ℓ output is $\begin{cases} 0 & \text{if } \ell > k \\ 1 & \text{if } \ell \leq k \end{cases}$.

The priority encoder gives the 3-bit address of the highest numbered comparator which was on.

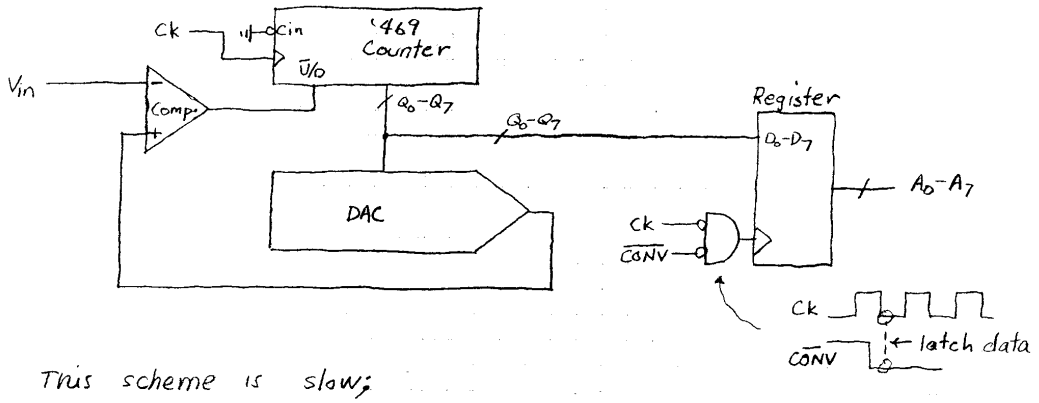
Note that the Register plays the role of a digital S & H; when CONV $\downarrow$ the comparator outputs are latched and V_{in} may change without affecting the conversion.

This is the fastest method of A-D conversion however typically it is limited to lower resolution (less bits) than other methods due to large amount of circuitry.

Also comparator V_{os} must be $< \underbrace{\frac{1}{2^m}(V_{RT} - V_{RB})}_{V_{LSB}}$ to avoid nonlinearity.

One can greatly reduce the number of Comparators etc needed with the addition of a DAC in the following feedback methods (4)

2. Tracking ADC



This scheme is slow;
the max $\frac{dV_{in}}{dt}$ that can be followed is $V_{LSB} * f_{ck}$.
If V_{in} changes faster than this then one should use a S&H in front of comparator and wait 256 (2^m) clock cycles.

3. Successive Approximation

This method iteratively bisects the voltage range to find the binary # of the interval containing V_{in}. At each iteration the range is reduced by 2. For m bits the answer is arrived at in m steps (rather than 2^m as above).

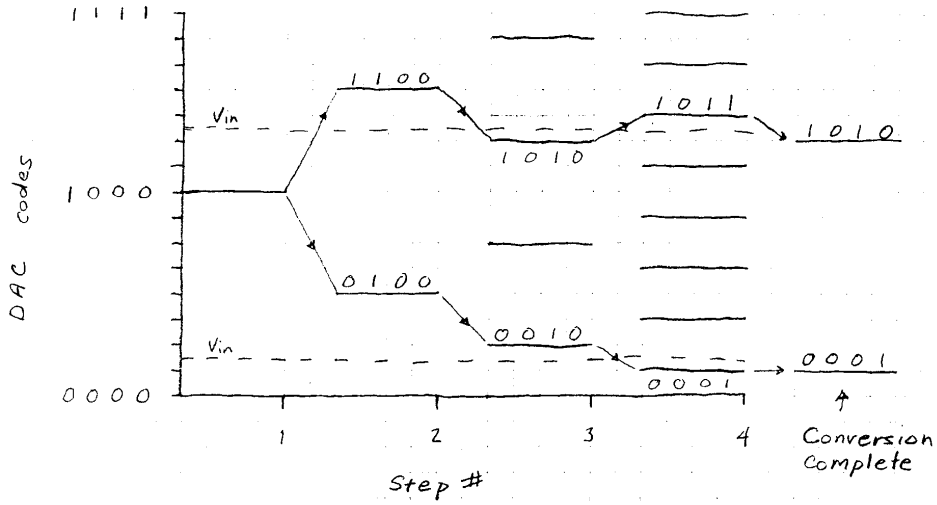
Step1: Set DAC MSB=1, all others =0 ie. 100... $\Rightarrow V_{DAC} = \frac{1}{2}$ F.S.

Step 2: if $\begin{cases} V_{in} > V_{DAC} & \text{leave MSB=1} \\ V_{in} < V_{DAC} & \text{set MSB=0} \end{cases}$ This redefines the range to be the upper or lower half of the original range. Then set bit MSB-1 = 1 and all lesser bits = 0,

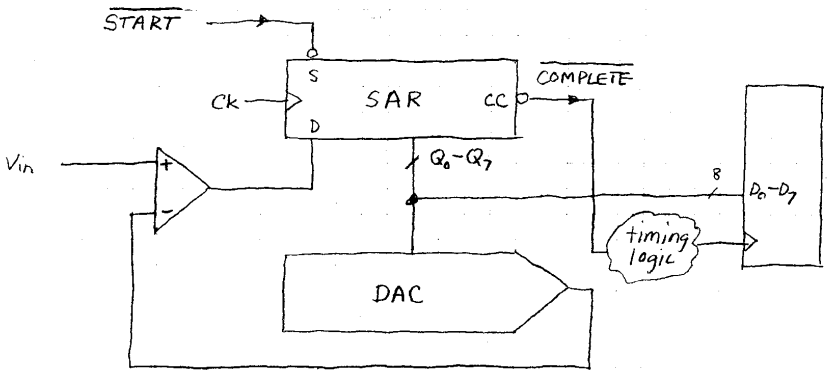
$$\Rightarrow V_{DAC} = \begin{cases} 3/4 \text{ F.S.} & \text{if } 1100\ldots \\ 1/4 \text{ F.S.} & \text{if } 0100\ldots \end{cases}$$

Step 3: Test $V_{in} \lessgtr V_{DAC}$ as above and adjust bit MSB-1. Then set bit MSB-2 =1 and all lesser bits = 0. etc.....

A graphical display of DAC in vs step # (time) is shown below for a 4-bit conversion and two different V_{in}



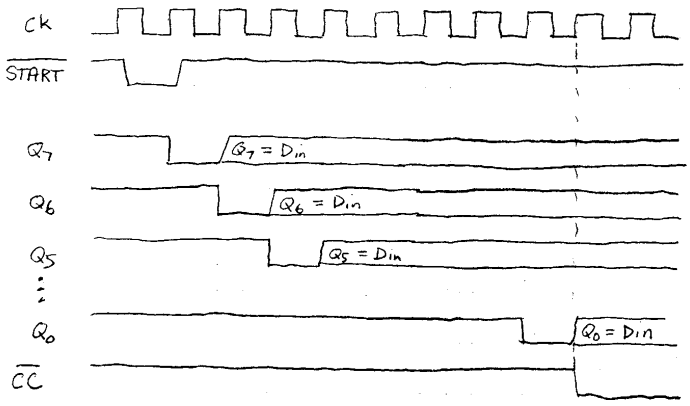
The sequential logic which does this is called a "Successive-Approximation Register" or SAR. Its use in an ADC is shown below.



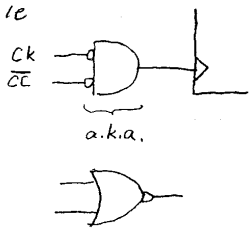
It is crucial that V_{in} remain constant within ½ LSB during the conversion. If this isn't the case then the comparator V_{in} input should be preceeded by a S&H

The SAR we use in Lab ('LS503) assumes neg-true logic on its I/O pins although it isn't obvious from the data sheet diagram. The LS503 assumes that a 0 data output adds current to the DAC output, ie 000... => F.S. and 111... => $V_{DAC}=0$. One can nevertheless hook it up as shown on prev. pg or in lab manual and it will work, but there will be a few missing codes even if DAC is perfect. Doing it correctly involves swapping +/- inputs of comparator and adding inverters between SAR outputs and DAC inputs, or using complementory current output if there is one.

Given the Ck and S inputs as shown the outputs are

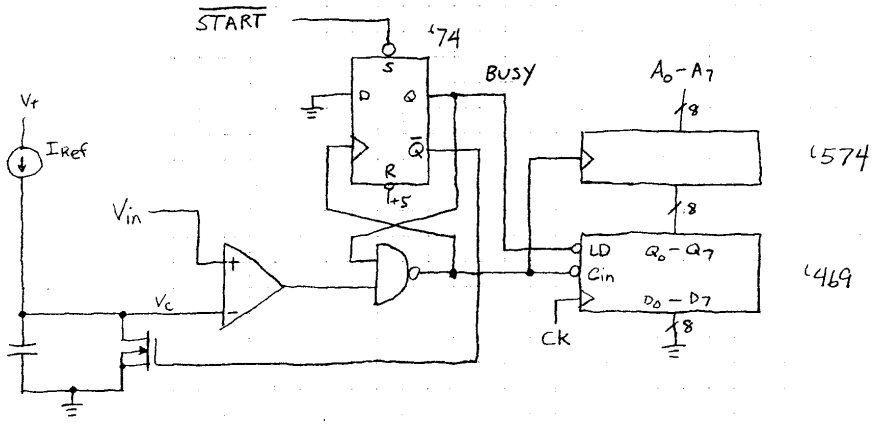


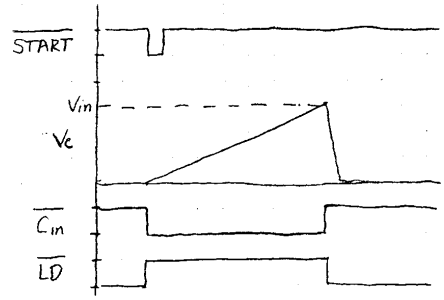
Since both Q_0 and $\overline{CC}$ change in response to Ck ↓ you would want to latch output data when Ck ↑ and $\overline{CC}=0$, ie



The linearity of the Tracking and SAR ADCs depends directly on the quality of the DAC, if it is nonlinear then the ADC will exhibit the same type of nonlinearity.
In the following ADCs V_{in} is converted into a pulse whose duration is measured with a stable clock and counter, avoiding the necessity of a DAC altogether.

4. Single Slope Converter





Ramp voltage $V_c(t) = \frac{I_{Ref}}{C} t$.
$\Rightarrow$ Duration $t = \frac{V_{in} C}{I_{Ref}}$.

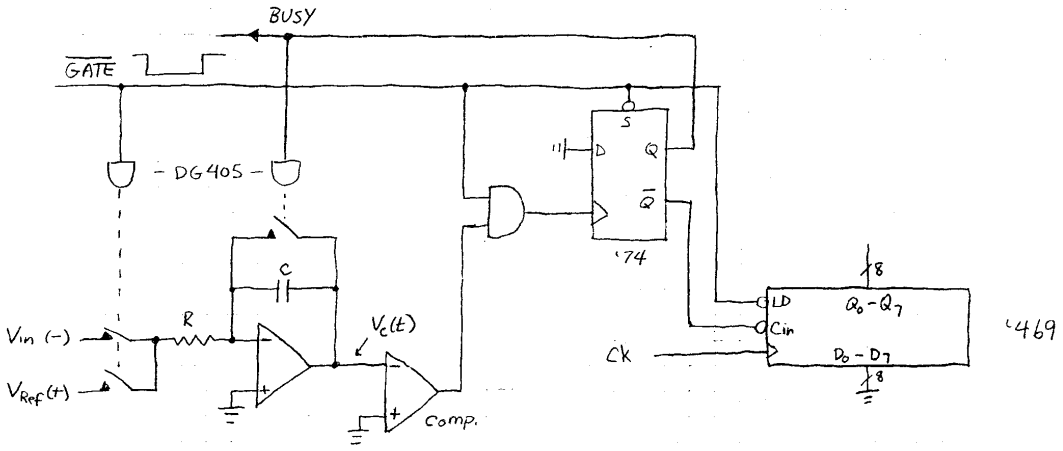
'469 counts while $\overline{C_{in}}$ asserted
load '469 to 00_{11} when $\overline{C_{in}}$ goes off

When done, the count $\boxed{m = f_{clock} \cdot t_{ramp} = \frac{f_{clock}\, C}{I_{REF}} \times V_{in}}$.

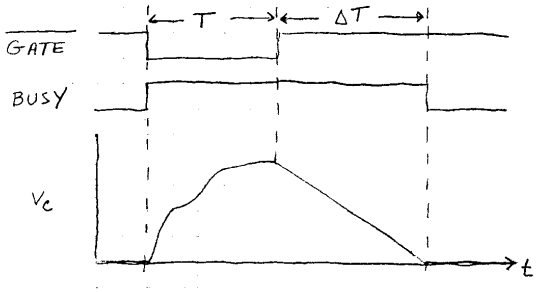
- ∃ scale errors if C, I_{REF} or f_{clock} drift w/ time & temp. However it is <u>monotonic</u> and DNL is good since any drift usually occurs on a much longer time scale than t_{ramp}.
- Any variation of V_{in} (e.g. noise spikes) occuring near end of ramp will cause error.

5. Dual Slope Converter

By averaging input this method is less sensitive to noise on V_{in}, and even becomes independent of R, C and perhaps f_{clock}.



Switch positions shown for $\overline{GATE}$ and BUSY true.



At the end of gate interval $t = T$;

$$V_c(T) = \frac{1}{RC}\int_0^T V_{in}\,dt = \frac{T}{RC}\langle V_{in}\rangle .$$

After gate ($T < t < T+\Delta T$) V_c "ramps-down" at rate $\dot{V}_c = -\frac{1}{RC}V_{Ref}$

$$\Rightarrow V_c(t) = V_c(T) - \frac{V_{ref}}{RC}(t - T)$$

The duration of the ramp-down measured by the counter is

$$\Delta T = \frac{RC}{V_{Ref}} \times \frac{T}{RC}\langle V_{in}\rangle = T\,\frac{\langle V_{in}\rangle}{V_{Ref}} \qquad (= m\tau_{clock})$$

Note - Result independent of RC

- If $T = n\tau_{clock}$ then result is also independent of f_{clock}!
  ie $m = n\,\frac{\langle V_{in}\rangle}{V_{Ref}}$ where m is measured ramp-down count.
- If T = integral # of 60Hz periods then result is not affected by 60Hz noise.

Of the various "slow" methods this one is the best in most cases.

(7)

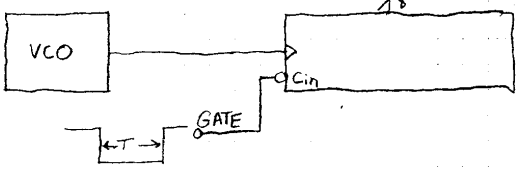
Other integrating converters ....

7. VFC (Voltage-to-Frequency)

This method counts the number of cycles of a voltage controlled oscillator occuring during a fixed time interval, say 1sec. Recall the VCO analyzed in Ex 4.13 on pg 240 of H.[2] Each cycle of the output waveform required a time τ given by

$$\frac{1}{R_1C_1}\int_0^{\tau} V_{in}\,dt = \frac{1}{3}V_+ \quad \Rightarrow \quad f = \frac{1}{\tau} = \frac{3}{R_1C_1}\frac{\langle V_{in}\rangle}{V_+}.$$

$$n = fT = \frac{3T}{R_1C_1}\frac{\langle V_{in}\rangle}{V_+}$$



8. Delta-Sigma converter

This is similar to the VFC except that neg feedback is use to insure a linear VCO.