CpE2210 Introduction to Digital Logic

Dr. Minsu Choi

CH 5: Digital Hardware & Logic Components

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

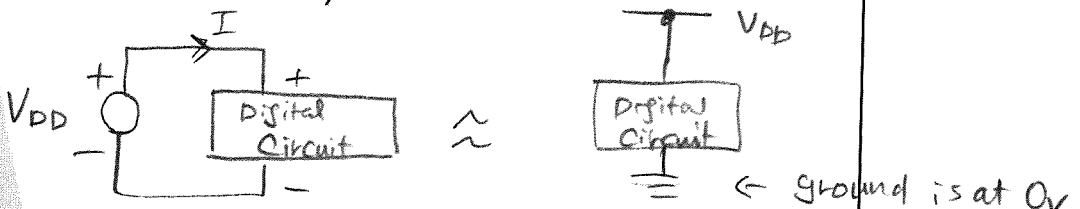
Hardware?

- The physical realization of a digital element or system.
- How to represent logic 0 and logic 1 states?
  1. Voltage V, which has units of volts (v).
  2. Electrical current I, which has units of ampres (A or amps).
- Two are related: a voltage causes electrical current to flow.
- Most digital logic chips use two different voltage ranges to define logic 0 and logic 1 conditions.

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Power Supply

- All electronic networks require a power supply to operate.
- In digital circuits, the power supply is usually modeled as a voltage source with a value that we will denote by V_{DD} (usually 5v, 3.3v or 2.5v).
- Ex) schematic diagram of a digital system with a power supply (since the drawing shows the "scheme" used to construct the circuit).

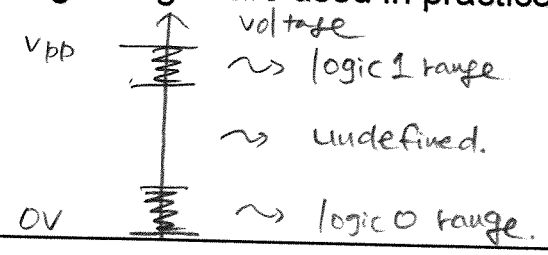


positive side voltage is higher as much as V_{DD} volts.

Logic Levels

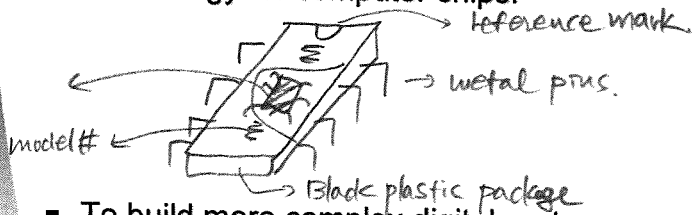
- Logic 0 -> 0v and Logic 1 -> V_{DD} (ex., 5v), in general.
  - Low voltage represents logic 0 & high voltage represents 1 -> called positive logic.
  - High voltage represents logic 0 & low voltage represents 1 -> called negative logic.
- Two voltage ranges are used in practice.

ex)



Digital Integrated Circuits

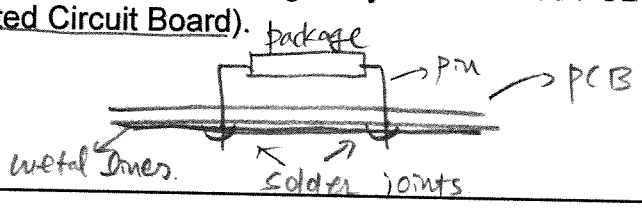
- ICs (Integrated Circuits) = Computer chips.
- Dual inline package (DIP) is very common packaging technology for computer chips.



Silicon chip inside

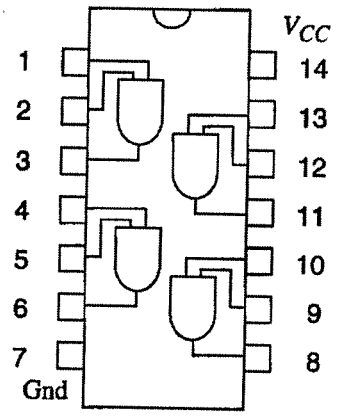
=) Complex circuit is fabricated on a Silicon chip by optical lithography.

- To build more complex digital systems -> use PCB (Printed Circuit Board).



Pin-Out Diagram

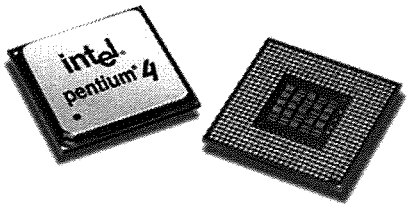
- The logic functions that a particular chip implements are usually shown by embedding equivalent logic diagrams in package outline drawings.
- Ex) Quad-AND chip. V_{CC} is an alternative notation for the power supply voltage. ≈ V_{DD}.



Ex) pin3 = pin1 • pin2.

Different Packaging Technologies?

- PGA (Pin Grid Array) is used for most microprocessor chips, since larger chip can be embedded and larger # of pins can be allocated.
- There are different variations – INTEL's LGA775 and AMD's Socket939.



ICs can be categorized with respect to the # of gates inside

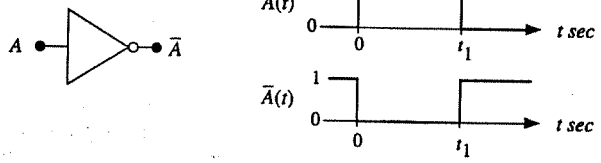
IC Integration Levels

- SSI (Small-Scale Integration): ICs with a few gates.
- MSI (Medium-Scale Integration): ICs with a few hundred gates.
- LSI (Large-Scale Integration): 1K – 100K gates.
- VLSI (Very Large-Scale Integration): a few million gates.
- ULSI (Ultra Large-Scale Integration): around one billion gates. six core i7
- Ex) INTEL ~~Pentium 4~~ processor has about ~~125 million~~ gates.

1.27 billion

Logic Delay Times

- Waveform (a plot of a variable as a function of time) is used to measure the behavior of a gate.
- Ex)



(a) Logic gate (b) Ideal waveforms

→ input waveform

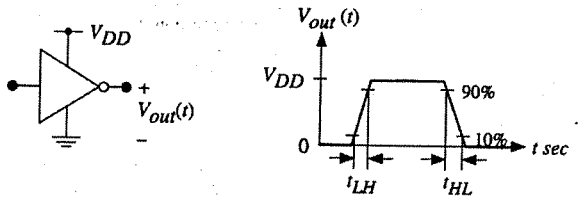
→ output waveform.

- In the real world, a voltage cannot chance instantaneously.

⇒ There should be some delays for 0→1 & 1→0 transitions

Output Switching Times

- Two time intervals must be considered:
  - t_{LH}, the output low-to-high time, also called the rise time, t_r.
  - t_{HL}, the output high-to-low time, also called the fall time, t_f.
  - By convention, these time intervals are not measured between 0v to VDD, but represent the transition required between 10% to 90% voltage levels, as shown below.



(a) Logic gate (b) Low-to-high and high-to-low times

Continued,

- The minimum amount of time needed for the gate to switch from 0 to 1 then back again is given by:

$$t_{min} = t_{LH} + t_{HL}$$

- The maximum switching frequency is:

$$f_{MAX} = \frac{1}{(t_{LH} + t_{HL})} \quad [\text{in Hz}]$$

- =max # of logic transitions that the gate can make in 1 sec.
- Ex) $t_{LH} = 7.2\,ns$ & $t_{HL} = 3.9\,ns$, f_{max}?

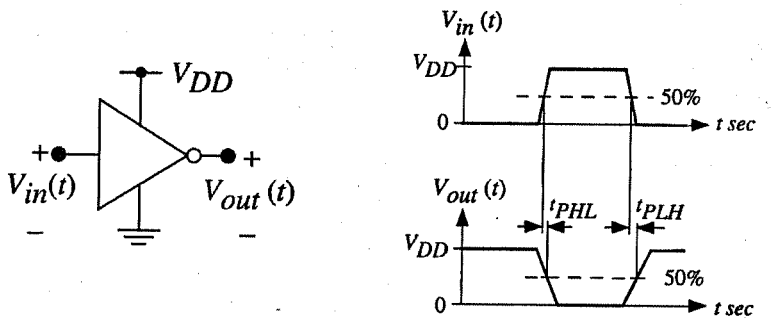
$$f_{max} = \frac{1}{7.2 \times 10^{-9} + 3.9 \times 10^{-9}} = 90.09\ MHz$$

(≈ 90.09 million transitions per sec max

Propagation Delay

- It can be hard to keep track of both delay times for every logic gate.
- At the logic design level, it is simpler to introduce a single delay time that represents an average switching time, called "propagation delay", from the input to the output.
- This is to include the physical delay of a logic signal as it "propagates" through a chain of gates.

Inverter Gate Example



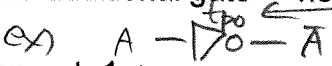
- Then, t_P (=propagation delay) = 1/2 (t_{PHL} + t_{PLH}) or =max(t_{PHL}, t_{PLH}), alternatively.

Fan-In & Fan-Out

- The fan-in of a digital logic gate refers to the number of inputs.
- Ex) inverter has fan-in of 1 and NAND2 has fan-in of 2.
- The fan-in provides information about intrinsic speed of the gate -> the propagation delay increases with the fan-in.
- Ex) OR2 is faster than OR3.

Continued,

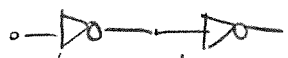
- The switching time of an electronic gate depends on the number of gates that are driven (or connected) at the output.
- The fan-out of a gate is the number of gates that are driven by the output, and it depends how the gate is used in the logic chain -> increasing the fan-out slows down the logic flow through the gate.
- If no additional gate -> no load (fan-out=0)



t_{p0} ← no load propagation delay.

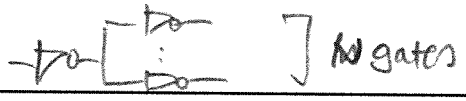
ex)

- Fan-out=1 ->



$t_{p1} = t_{p0} + 1 \cdot t_{PL}$ ← additional delay time needed to drive the load (next inverter).

- Fan-out=N ->



$$t_{pN} = t_{p0} + N \cdot t_{PL}$$

Logic Cascades and Delays

- Ex) linear chain of 4 inverters.

$t_d = t_{d1} + t_{d2} + t_{d3} + t_{d4}$

Fan-in & fan-out delays must be considered.

$t_{d4} = t_{to,NOT} + t_L$

$t_{to,NOT}$: no-load P.D.

t_L: The load at the forth gate is not specified. So, t_L is used.

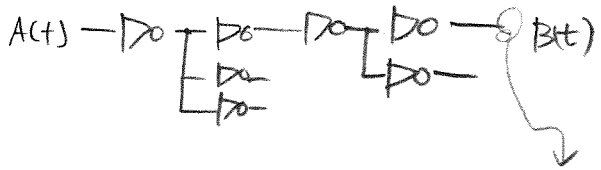
$t_d = 4 \cdot t_{to,NOT} + 3\, t_{PL,NOT} + t_L$

$4 \cdot t_{to,NOT}$: no-load prop delay

$3\, t_{PL,NOT}$: fan-out delay

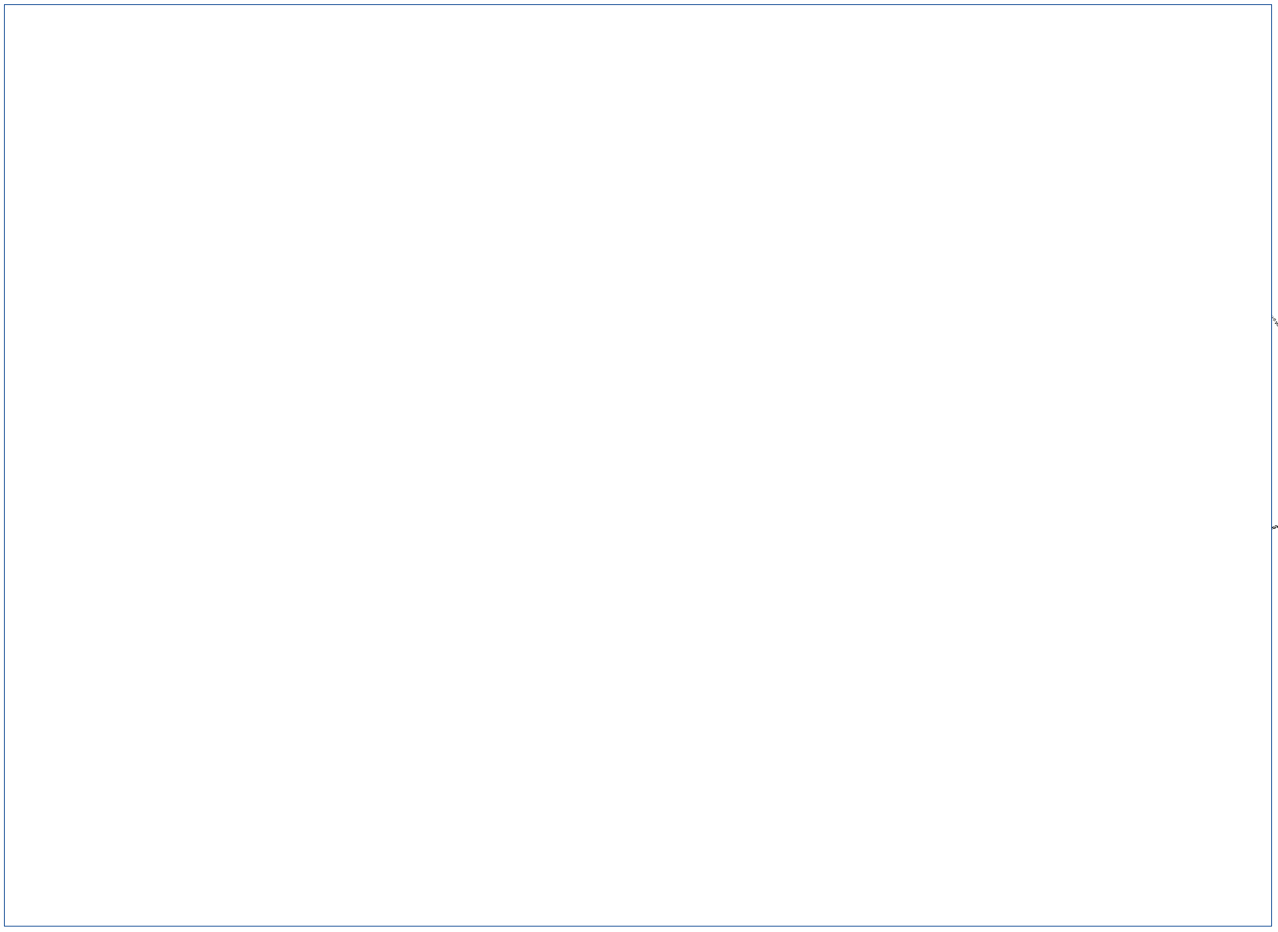
Continued,

- Ex) Inverter chain with increased internal delay.



$$t'_d = \underbrace{4 \cdot t_{PO,NOT}}_{\text{no-load prop delay}} + \underbrace{6 \cdot t_{PL,NOT}}_{\text{6 fan-outs driven}} + t_L$$

$\Rightarrow$ takes more time because of more fan-outs!

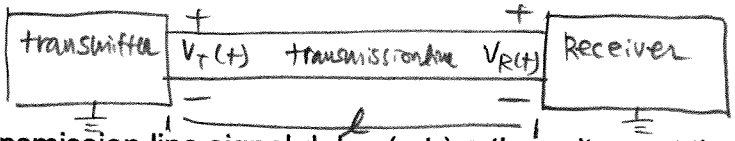


Application to Digital Circuits

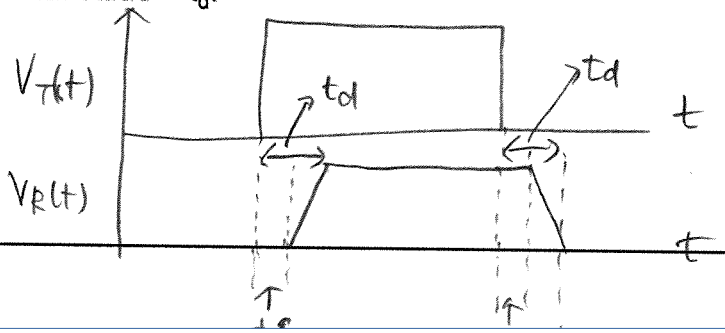
- Transitions 0->1 and 1->0 take time.
- The switching time of every digital electronic network is governed by any resistance and capacitance in the network.
- Parasitic elements: unwanted resistance and capacitance contributions that cannot be eliminated and that act to slow down the network response.
- So, keeping R and C as small as possible is desired!

Transmission Lines

- Another type of logic delay arises when we analyze the physics of transmitting a voltage along a wire.



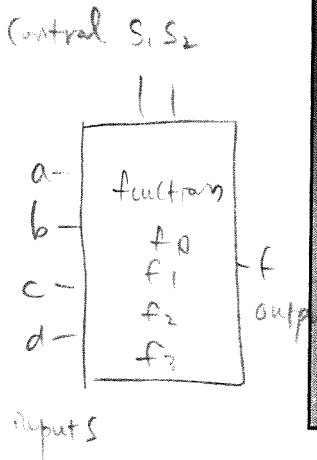
- Transmission line signal delay (= t_s) + the voltage at the end of transmission line takes time to "build up" to the final value = t_d.



Digital Logic Component Concept

- It is very difficult to keep track of each gate in designing a large digital system such as a computer.
- The hierarchical design approach where a large ("macro") function is defined using a large block is called a digital component (= element, unit or module).
- Ex)

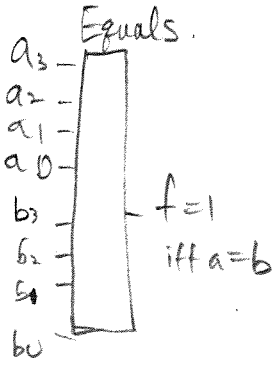
$f = f_0\ (a, b, c, d)$ if $S_1 S_0 = 00$
" 1 " 01
" 2 " 10
" 3 " 11



Control bits select internal functions.

XNOR is the equivalence function.

Block-diagram

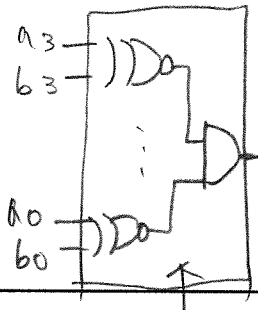


4-Bit Equality Detector

- Suppose that we have two 4-bit words
  $a = a_3 a_2 a_1 a_0$ $b = b_3 b_2 b_1 b_0$
- That we wish to compare.

if (a=b) then f=1 else f=0

So, $f = (\overline{a_3 \oplus b_3}) \cdot (\overline{a_2 \oplus b_2}) \cdots (\overline{a_0 \oplus b_0})$

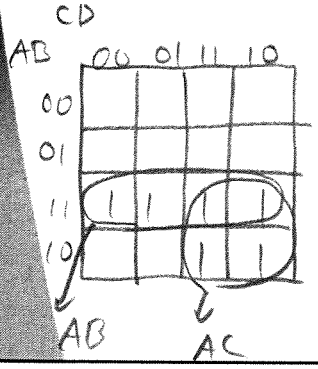


f = 1 iff a = b.

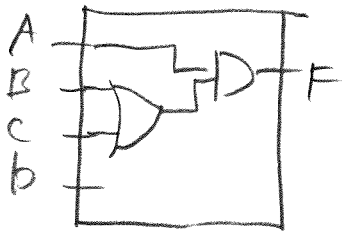
internal circuitry.

BCD Validity Detector

- Bit patterns from 0000 to 1001 are used.
- How can we design a detector?
- If (ABCD is valid) then F=0 else F=1.
- Let's build a K-map!

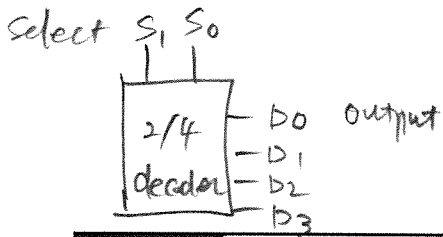


$F = AB + AC = A(B + C)$



BCD Validation Unit.

b is "don't care"



F.T.

S_1 S_0	D_3	D_2	D_1	D_0
0 0	0	0	0	1
0 1	0	0	1	0
1 0	0	1	0	0
1 1	1	0	0	0

↳ 2-bit binary word selects the output port to be activated ⇒ called Active-high decoder.

Line Decoders

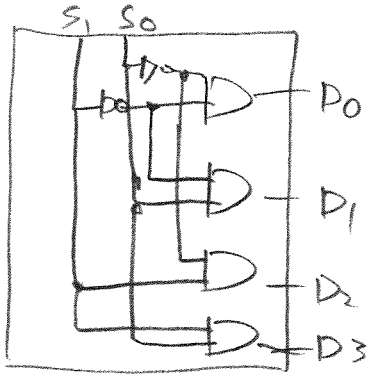
- Line decoder is a circuit that allows us to "activate" an output line by specifying a control word.
- Ex) Active-high 2/4 decoder

$D_0 = \overline{S_1} \cdot \overline{S_0}$

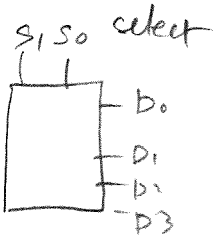
$D_1 = \overline{S_1} \cdot S_0$ ⇒

$D_2 = S_1 \cdot \overline{S_0}$

$D_3 = S_1 \cdot S_0$



⇐ internal circuitry for the 2/4 decoder.



S_1 S_0	D_3	D_2	D_1	D_0
0 0	1	1	1	0
0 1	1	1	0	1
1 0	1	0	1	1
1 1	0	1	1	1

Active-Low 2/4 Decoder

$D_0 = \overline{\overline{S_1} \cdot \overline{S_0}}$

$D_1 = \overline{\overline{S_1} \cdot S_0}$ ⇒ replace AND gates of active-high decoder to NAND gates.

$D_2 = \overline{S_1 \cdot \overline{S_0}}$

$D_3 = \overline{S_1 \cdot S_0}$

or, by DeMorgan's rule...

$D_0 = S_1 + S_0$

$D_1 = S_1 + \overline{S_0}$ ⇒ NOT & OR gates can be used.

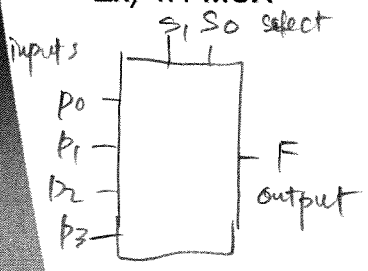
$D_2 = \overline{S_1} + D_0$

$D_3 = \overline{S_1} + \overline{S_0}$

⇒ Either way, the external behavior remains the same regardless of the internal circuit details.

Multiplexer (MUX)

- A MUX is a logic component that has several inputs but only a single output. It provides the capability to direct one of the inputs to the outputs.
- Ex) 4:1 MUX

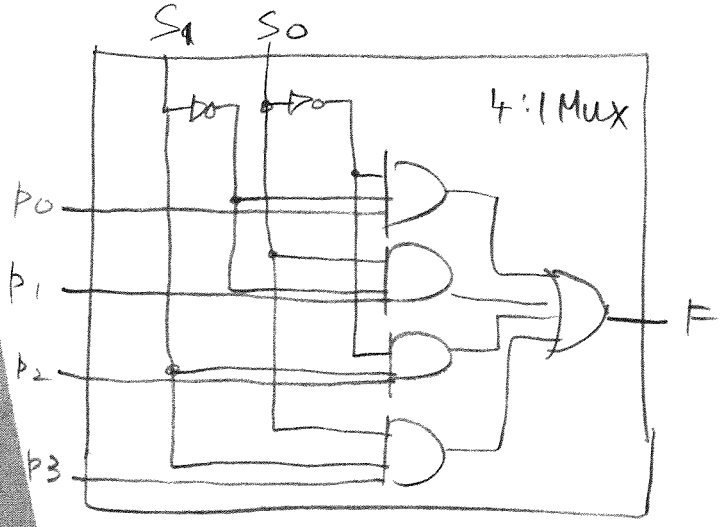


S_1	S_0	F
0	0	P_0
0	1	P_1
1	0	P_2
1	1	P_3

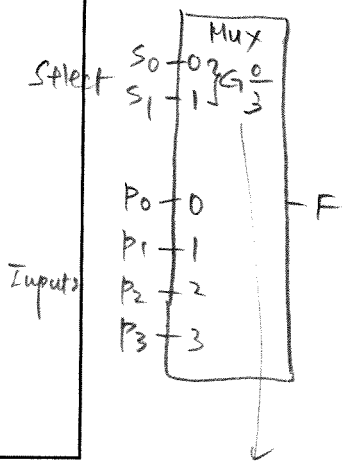
SOP form

$F = \bar{S_1}\bar{S_0} \cdot P_0 + \bar{S_1} S_0 \cdot P_1 + S_1 \bar{S_0} P + S_1 S_0 P_3$

4:1 MUX: Internal Circuitry and IEEE Symbol

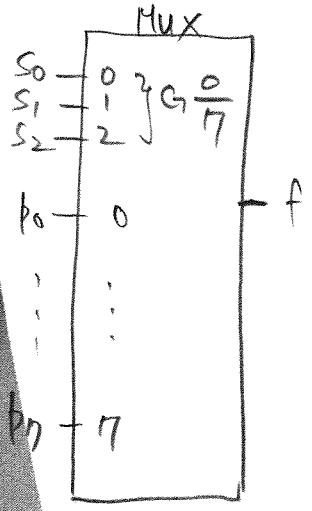


IEEE symbol



G-dependance notation
(select bits S_1 & S_0 select input 4 signal $P_0 \sim P_3$)

8:1 MUX



- Internal circuitry will be very complex.
- 2 4:1 MUXs & 1 2:1 MUX can be combined to implement 8:1 MUX.
- -> Hierarchical design approach.

Continued,

- The select word $s_2s_1s_0$ can be split into two groups: s_1s_0 to control the 4:1 MUXs and s_2 to select a desired output from them.
- Two internal signal x_1 and x_2 can be defined as:

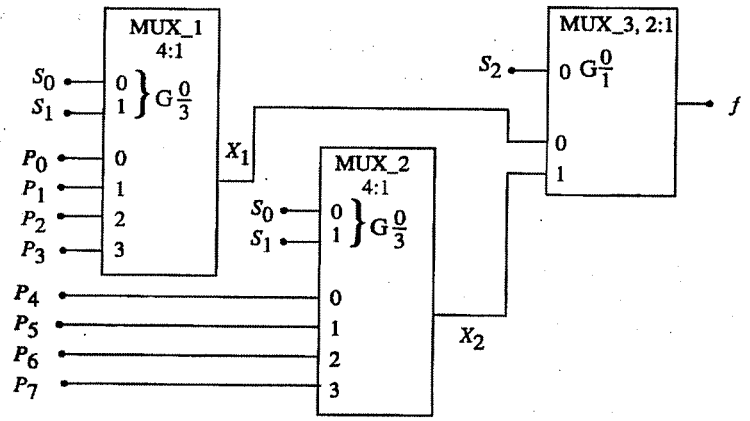
$$x_1 = \bar{s}_1\bar{s}_0P_0 + \bar{s}_1s_0P_1 + s_1\bar{s}_0P_2 + s_1s_0P_3$$

$$x_2 = \text{"}\ P_4 \quad \text{"}\ P_5 \quad \text{"}\ P_6 \quad \text{"}\ P_7$$

- Then,

$$f = \bar{s}_2 \cdot x_1 + s_2 \cdot x_2$$

Completed 8:1 MUX Design



MUXs as Logic Elements

- A MUX can be programmed to function as a logic element.
- Ex) 4:1 MUX OR implementation.

The output equation of 4:1 MUX is...

$f(x,y) = \bar{x}\bar{y}P_0 + \bar{x}yP_1 + x\bar{y}P_2 + xyP_3$

T.T of OR is

x	y	f
0	0	0
0	1	1
1	0	1
1	1	1

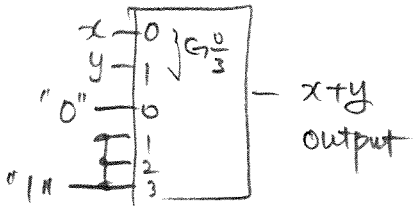
replace $P_0 = 0$, $P_1 = 1$, $P_2 = 1$, $P_3 = 1$

$f = \bar{x}y + x\bar{y} + xy$

↙ K-map reduction

x \ y	0	1
0	0	1
1	1	1

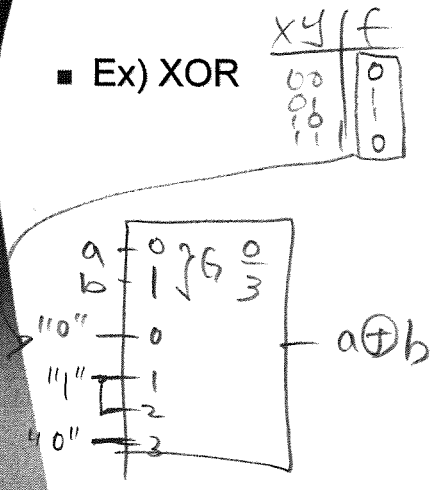
$f = x + y$



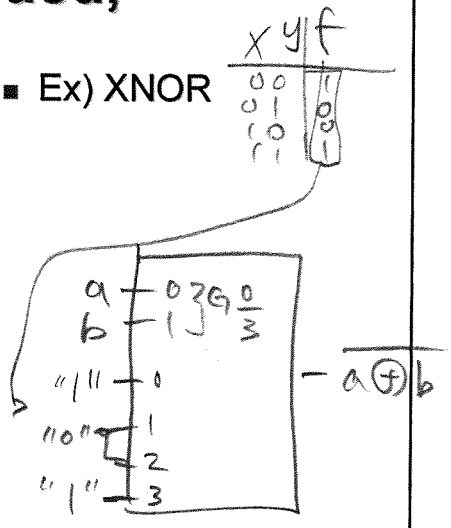
⊛ Input vars → control bits of MUX

Continued,

- Ex) XOR



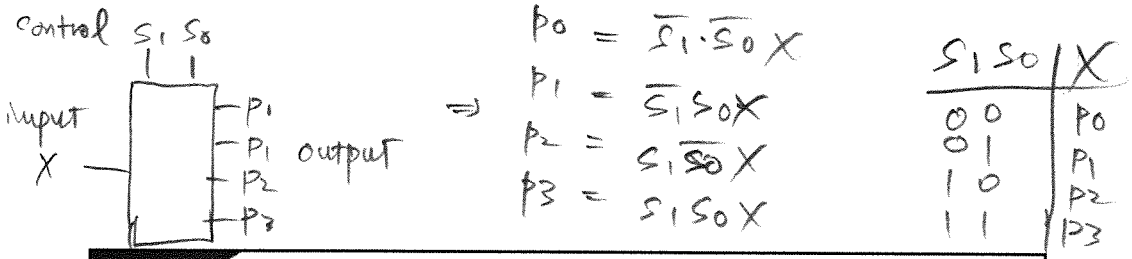
- Ex) XNOR



VHDL Description of 4:1 MUX

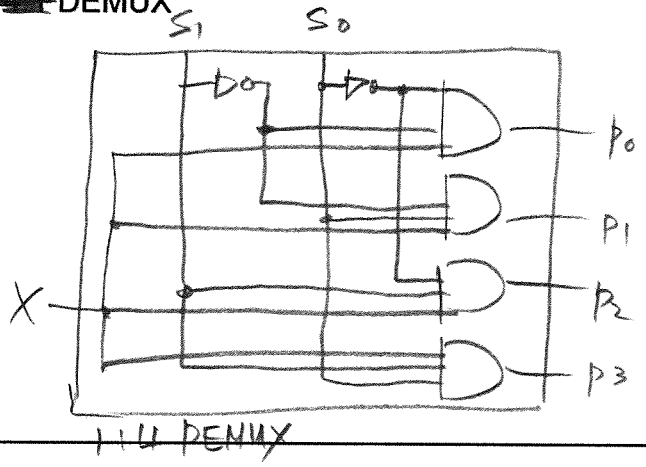
```
entity mux4 is
   port(d0, d1, d2, d3: in
   bit; s : in bit_vector(1
   downto 0); f : out bit);
end mux4;

architecture basic of
   mux4 is
begin
   f<=d0 when (s="00")
   else
   f<=d1 when (s="01")
   else
   f<=d2 when (s="10")
   else
   f<=d3 when (s="11")
end basic;
```



Demultiplexer (DEMUX)

- A DEMUX is the opposite of a MUX. It takes a single input and directs it to one of several outputs.
- Ex) 1:4 DEMUX



VHDL Code

S: in bit_vector(1 downto 0);

```
entity demux1_4 is
   port(x: in bit;
   p0,p1,p2,p3 : out bit);
end demux1_4;

architecture operation of
   demux1_4 is
```

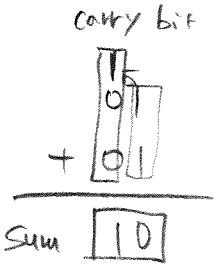
```
begin
   ifs="00" then
        p0<=x;
        p1<='0';
        p2<='0';
        p3<='0';
   else if s="01" then
...
end operation;
```

Binary Adders

- Arithmetic functions such as addition and subtraction can be performed using binary numbers. These types of operations are central to building a computer.
- 4 cases can be identified for 1-bit addition:
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 0 with a carry of 1 (called "carry bit")
- The output column is equivalent to XOR function.

⇒ XOR

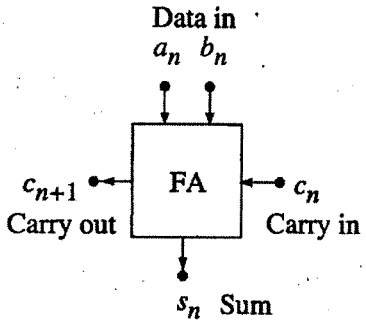
ex)



Full Adder

- A logic network that provides the operations needed to add the bits in an arbitrary column.
- 3 input bits: a_n, b_n (data bits) and c_n (carry-in bit from the column immediately to the right).
- 2 output bits: s_n (sum bit) and c_{n+1} (carry-out bit).

Block Diagram and Truth Table



(a) Block diagram

a_n	b_n	c_n	s_n	c_{n+1}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

(b) Function table

Simplified Function Forms of s_n and c_{n+1}

⇒ K-maps proven not to be effective.

$S_n = \bar{a}_n b_n \bar{c}_n + a_n \bar{b}_n \bar{c}_n + \bar{a}_n \bar{b}_n c_n + a_n b_n c_n$

$= (\bar{a}_n b_n + a_n \bar{b}_n)\bar{c}_n + (\bar{a}_n \bar{b}_n + a_n b_n) c_n$

$= (a_n \oplus b_n)\bar{c}_n + \overline{(a_n \oplus b_n)} c_n$

$\boxed{= a_n \oplus b_n \oplus c_n}$ <= Odd function.

$C_{n+1} = a_n b_n \bar{c}_n + \bar{a}_n b_n c_n + a_n \bar{b}_n c_n + a_n b_n c_n$

$= a_n b_n (\bar{c}_n + c_n) + c_n (\bar{a}_n b_n + a_n \bar{b}_n)$

$\boxed{= a_n b_n + c_n (a_n \oplus b_n)}$

Half Adder

- A special case of the full adder when $c_n=0$ (no carry-in bit considered).

Data in

a_n b_n

C_{n+1} ← HA

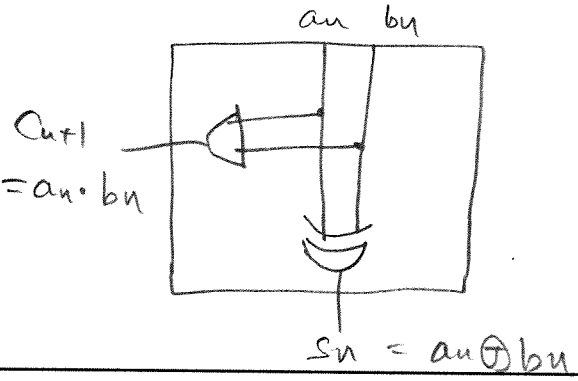
Carry out

↓ S_n sum

a_n	b_n	S_n	C_{n+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S_0, $S_n = a_n \oplus b_n$, $C_{n+1} = a_n \cdot b_n$

⇒

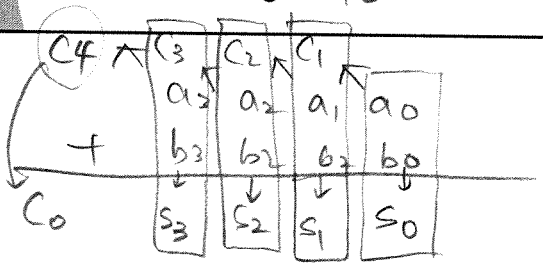


Parallel Adders

- Half-adders and full-adders are used to add individual bits together. An extension of this problem is the addition of two n-bit binary words.
- Ex) $A = a_3a_2a_1a_0$ & $B = b_3b_2b_1b_0$ ⇒ 4-bit adder

Let's break down the problem explicitly by writing the addition procedure out as:

- This shows that we can use a single-bit adder for each column and connect the carry-out bit to the adjacent column to the ~~right~~ left.



lowercase "0" overflow bit.

Overflow Bit

- Also note that the sum may be larger than 4 bits.
- So, an overflow bit c_o is provided such that c_o =0 means that 4 bits are sufficient to express the sum, while c_o =1 implies that the sum requires more than 4 bits.
- Ex)

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

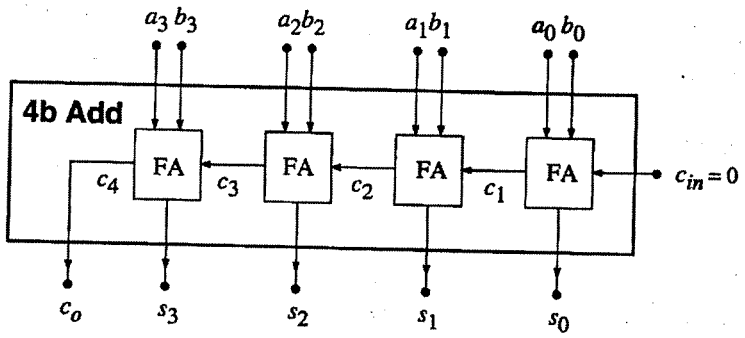
no overflow

overflow

⇒ now the result requires 5 bits (overflow). This is because the answer (27_{10}) is larger than the max value (15_{10}) that can be represented by 4-bit word!

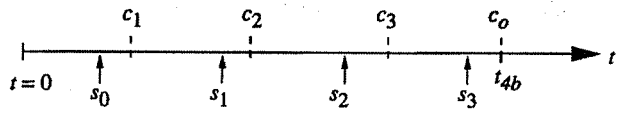
Parallel Adder Design with Ripple Carry Scheme

- The carry "ripples" from the right to the left.

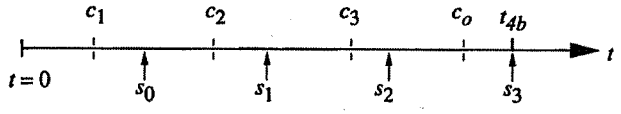


Timing

- There are two cases we must consider:
  - FA generates s first, then c.
  - FA generates c first, then s.

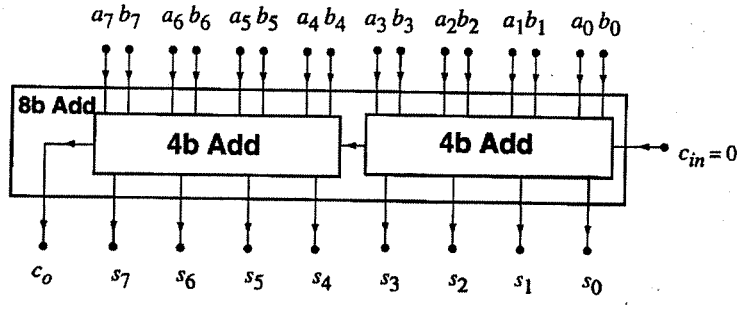


(a) FA produces sum bit first



(b) FA produces carry-out bit first

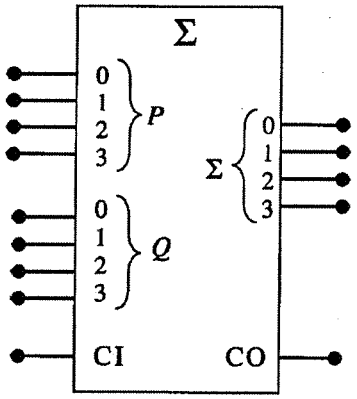
8-Bit Adder Design using Cascaded 4-Bit Adders



- Likewise, 16-bit or more can be designed.

IEEE Adder Symbol

ex) 4-bit adder



- P: input word 1
- Q: input word 2
- Sigma: output word
- CI: carry-in bit
- CO: carry-out bit

Subtraction

- It is more complex since we need to "borrow".
- Ex)

$0 - 0 = 0$

$1 - 0 = 1$

$1 - 1 = 0$

$\boxed{0 - 1 = 1}$ (using a borrow of 1 from the left)

ex)

$$\begin{array}{r} {\scriptstyle \text{"borrow"}\ \curvearrowright} \\ 1\ 0 \\ -\ 0\ 1 \\ \hline 0\ 1 \end{array}$$

- Very hard to implement with digital network.

More Efficient Way?

- X – Y = D is equivalent to saying that X + (-Y) = D where Y + (-Y) = 0.
- So, we can find (-Y). Then, using the adder, add X & (-Y) to perform subtraction.
- Ex) $Y = y_3 y_2 y_1 y_0$ $(-Y) = w_3 w_2 w_1 w_0$

then $Y + (-Y) = 0 \Rightarrow$

$$\begin{array}{r} y_3\ y_2\ y_1\ y_0 \\ +\ w_3\ w_2\ w_1\ w_0 \\ \hline 0\ 0\ 0\ 0 \end{array}$$

If we choose $w_n = \overline{y_n}$, then the sum in every column is automatically $(1+0)=1$, so that:

$$\begin{array}{r} y_3\ y_2\ y_1\ y_0 \\ +\ \overline{y_3}\ \overline{y_2}\ \overline{y_1}\ \overline{y_0} \\ \hline 1\ 1\ 1\ 1 \end{array}$$ ← should be 0000

Idea!: add 1 to make it 0000.

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ +\ 1 \\ \hline \boxed{1}\ \boxed{0\ 0\ 0\ 0} \end{array}$$

all 0s

⇒ the goal accomplished

ignore carry.

1's Complement and 2's Complement

- Then, how we can express (-Y) in binary?
- 2's complement number can be used.
  - First, complement each bit: called 1's complement.
  - Then, add 1 to it: called 2's complement.
  - Y + 2's complement Y = 0, if we discard overflow bit.
  - So, (X – Y) = X + 2's complement Y -> we can use the adder to perform subtraction.
- Ex) X = 0101 and Y = 0011. Then, (X – Y) = ?

① 1's complement $(Y) = 1100$

② 2's " $(Y) = 1100 + 1 = 1101$

③ add them up.

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 1\ 1\ 0\ 1 \\ \hline \boxed{1}\ \boxed{0\ 0\ 1\ 0} \end{array}$$

discard ← ① → result.

$X = 5_{10}$

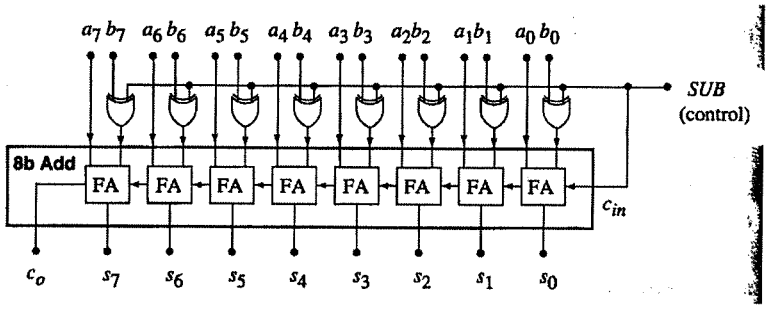
$Y = 3_{10}$

$X - Y = 2_{10}$.

Subtraction Logic Circuit

- XOR gate can be used to generate 1's complement number.
- C_{in} must be 1 so that we add 1 to 1's complement to make 2's complement.

SUB	b_n	$f = SUB \oplus b_n$	
0	0	0	$= b_n$
0	1	1	
1	0	1	$= \overline{b}_n$
1	1	0	



Positive and Negative Integers in 2's Complement

- n-bit binary word has 2^n bit patterns.
- The total number of bit patterns is divided into two groups.
- If MSB is 0, positive integer.
- If MSB is 1, negative integer.
- So MSB is also called as "sign bit".
- Since all-zero bit pattern is used to represent 0, the range of positive values is one less than the range of negative values.
- For example, 8-bit 2's complement word has 256 bit patterns. So, it is possible to express -128_{10} to $+127_{10}$.

S	a_6	a_5	a_4	a_3	a_2	a_1	a_0
-128	64	32	16	8	4	2	1

8-Bit 2's Complement Number

- The MSB is used as sign bit of -2^7 weight and the other bits have weight of 2^i.

$A = \boxed{S}\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$

S: sign bit of -2^7 weight $= S \times (-2^7)$

$a_6 a_5 a_4 a_3 a_2 a_1 a_0$: positive bin #

- Ex) $0000\ 0001 = 1_{10}$

$1000\ 0001 = -128_{10} + 1_{10} = -127_{10}$

$\vdots$

$1111\ 1111 = -128_{10} + (64_{10} + 32_{10} \cdots + 1_{10}) = -1_{10}$

Multiplication

- AND function can be used to implement 1-bit multiplication.

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$
$\Rightarrow$ AND function

bit-multiplicator.

- For multi-bit multiplication, 1-bit multiplier and HA can be used.
- Ex) 2-bit multiplication

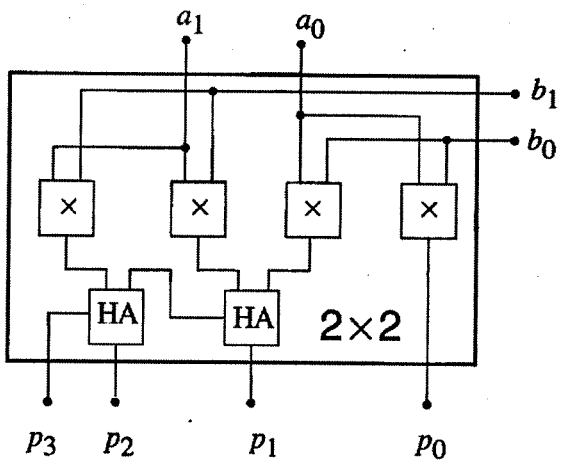
$$\begin{array}{cccc} & & a_1 & a_0 \\ & & b_1 & b_0 \\ \hline & & a_1b_0 & a_0b_0 \\ & a_1b_1 & a_0b_1 & \\ \hline P_3 & P_2 & P_1 & P_0 \end{array}$$

Product $\Rightarrow P_3\ P_2\ P_1\ P_0$

ex)

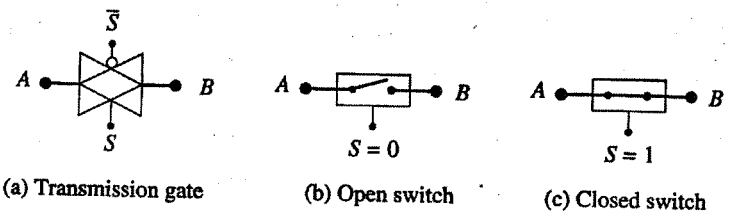
$$\begin{array}{r} 11 \\ \times\ 10 \\ \hline 00 \\ 11 \\ \hline 110 \end{array}$$

2-Bit Multiplier Internal Circuitry

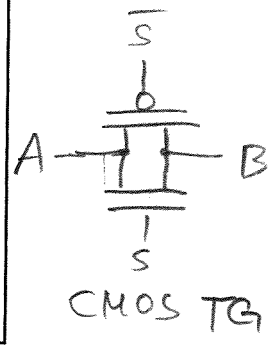


Transmission Logic Gate

- TGs are logic-controlled switches that can be used to construct a wide variety of logic networks.
- CMOS TG has very simple structure:



(a) Transmission gate (b) Open switch (c) Closed switch

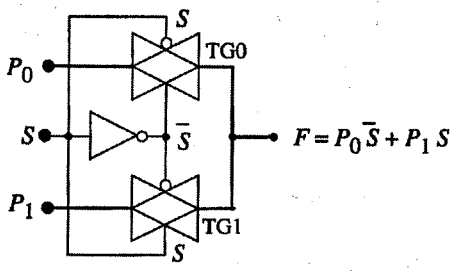


CMOS TG

If S=0, then open. (B is not determined by A)

If S=1, then closed (B=A)

2:1 MUX using TGs

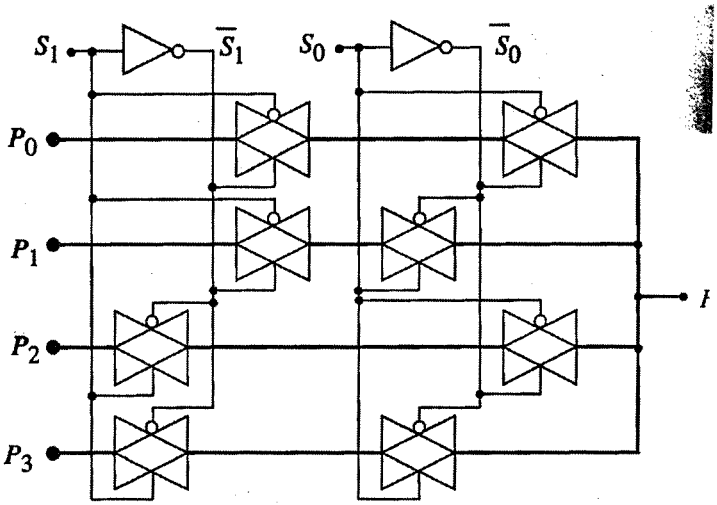


(a) Logic diagram

S	F
0	P_0
1	P_1

(b) Function table

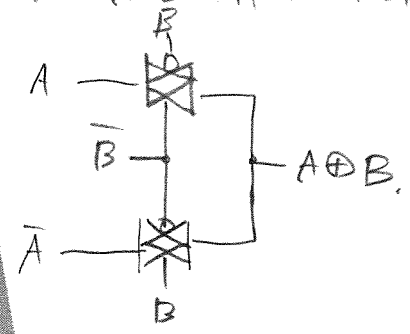
4:1 MUX using TGs



TG XOR and XNOR Gates

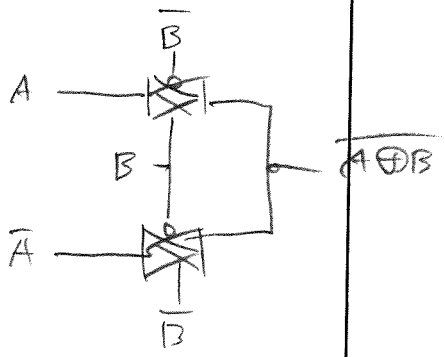
Ex) XOR

$F = A \cdot \bar{B} + \bar{A} \cdot B = A \oplus B$



Ex) XNOR

$F = A \cdot B + \bar{A} \cdot \bar{B} = \overline{A \oplus B}$



Program Completed

University of Missouri-Rolla

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.