

Contents

1 Gates, Expressions, Circuits, and Analysis 1/13 and 1/15	2
1.1 Logic Gates	2
1.2 Boolean Algebra	4
1.3 Combinational Logic Circuits	6
1.4 Standard Design Approach Sum of Products (SOP)	8
1.5 Karnaugh Map	9
2 CMOS Gate Design and Analysis 1/27	10
2.1 Metal Oxide Semiconductor (MOS) Transistor	11
3 CMOS oscillator (clock) properties and design 1/29	12
3.1 Relationship between Voltage and Logic	14
3.2 Clock and Clock design	15
3.3 Clock design	17
4 Common components and analysis 2/3	19
4.1 Multiplexer (MUX)	21
4.2 Demultiplexer (DeMUX)	23
4.3 Decoder	24
4.4 Encoder	25
5 ALU component design and analysis 2/5	27
5.1 Add and Subtract	27
5.2 FA Component Properties	30
5.3 Add and subtract Circuit	31
5.4 Bit Shift Circuit	33
6 Latches, D Flip-Flops (introduce J/K and T Flip-Flops), and memory/register design 2/24	35
6.1 Latch Component	35

1 Gates, Expressions, Circuits, and Analysis

1/13 and 1/15

Topics:

- Digital Logic Gates
- Boolean Algebra
- Combination Logic Circuits
- Sum of Products
- Karnaugh Maps

1.1 Logic Gates

A gate has (for example NOT gate):

1. Name
2. Schematic Diagram
 - Input, A, for example, with boolean (0 or 1)
 - Output, Y, for example, boolean (0 or 1)
3. Boolean Expressions, i.e. $Y = \overline{A}$
4. Truth Table

Example

We can also have two or more input gates:

- AND $\rightarrow Y = AB$, A and B must be true
- OR $\rightarrow Y = A + B$, A or B must be true
- XOR $\rightarrow Y = A \oplus B$
- NAND $\rightarrow Y = \overline{AB}$
- NOR $\rightarrow Y = \overline{A+B}$
- XNOR $\rightarrow Y = \overline{A \oplus B}$

A nice to know is that if the NOT's are the actual gate, then it would turn, for example, $X = \overline{A} \overline{B} \neq X = \overline{AB}$.

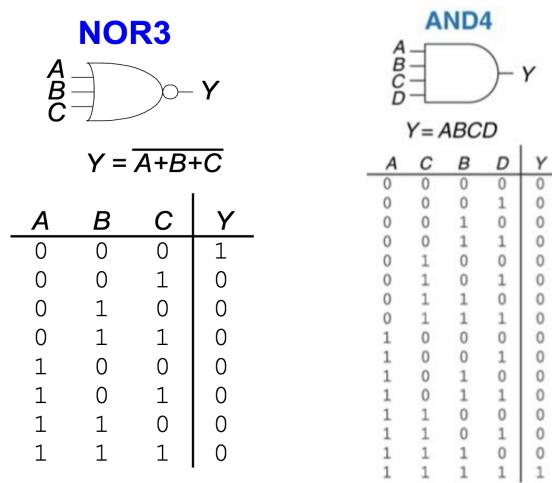


Figure 1: It can also have more than 2 inputs as seen here with their truth tables

1.2 Boolean Algebra

Symbols and Boolean operators:

$x \cdot y$, xy , $x \wedge y$, AND(x,y), x AND y

$x + y$, $x \vee y$, OR(x,y), x OR y

\bar{x} , x' , $\neg x$, NOT(x), INV(x)

$\overline{x \cdot y}$, $\overline{x \wedge y}$, \overline{xy} , NAND(x,y), x NAND y

$\overline{x + y}$, $\overline{x \vee y}$, NOR(x,y), x NOR y

$x \oplus y$, XOR(x,y), x XOR y

$x \overline{\oplus} y$, $\overline{x \oplus y}$, XNOR(x,y), x XNOR y

Figure 2: Notation before we get started

Moreover, here are some basic identities of boolean algebra

Basic Identities of Boolean Algebra

1.	$X + 0 = X$	2.	$X \cdot 1 = X$	
3.	$X + 1 = 1$	4.	$X \cdot 0 = 0$	
5.	$X + X = X$	6.	$X \cdot X = X$	
7.	$\overline{X + \overline{X}} = 1$	8.	$X \cdot \overline{X} = 0$	
9.	$\overline{\overline{X}} = X$			
10.	$X + Y = Y + X$	11.	$XY = YX$	Commutative
12.	$X + (Y + Z) = (X + Y) + Z$	13.	$X(YZ) = (XY)Z$	Associative
14.	$X(Y + Z) = XY + XZ$	15.	$X + YZ = (X + Y)(X + Z)$	Distributive
16.	$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	17.	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$	DeMorgan's

Figure 3: Some basic identities

Definition

Variable Substitution, is a way of substitution that makes it more tangible and math more easy

$$\begin{aligned} \mathbf{ABC + YZ = (ABC + Y)(ABC + Z)} \\ \text{Substitute } X \text{ for } ABC \\ \mathbf{X + YZ = (X+Y)(X+Z)} \end{aligned}$$

DeMorgan's Identity is used a lot and is very useful. As shown in these examples:

Example

<p>16. $\overline{X+Y} = \overline{X} \cdot \overline{Y}$</p> <p>NOR</p> <p>$A \cdot \overline{B} = \overline{A} \cdot B$</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0	<p>17. $\overline{X \cdot Y} = \overline{X} + \overline{Y}$</p> <p>NAND</p> <p>$A + \overline{B} = \overline{A} + B$</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0
A	B	Y																													
0	0	1																													
0	1	0																													
1	0	0																													
1	1	0																													
A	B	Y																													
0	0	1																													
0	1	1																													
1	0	1																													
1	1	0																													

Where the two equivalencies share a truth table due to this Identity

We can also see that it is like "pushing the bubble" as seen in this example:

- imagine the bubble at the output is being pushed towards the inputs
 1. it becomes a bubble at every input, and
 2. the shape of the gate changes from AND to OR, and vice versa



1.3 Combinational Logic Circuits

Definition

Stateless Digital Logic Circuits:

- Combinational logic combination of logic gates
- Change input values
- Immediate change in output values
- No Memory
- No feedback

Remark 1. There are specifics types of wire connections

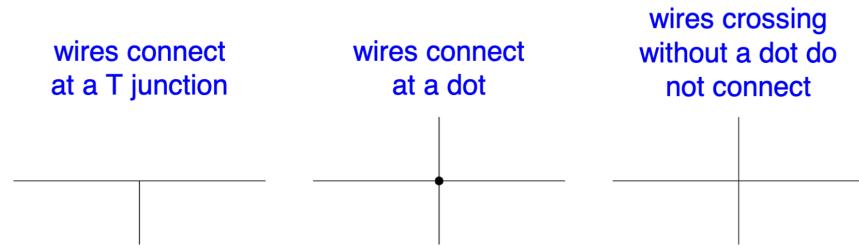
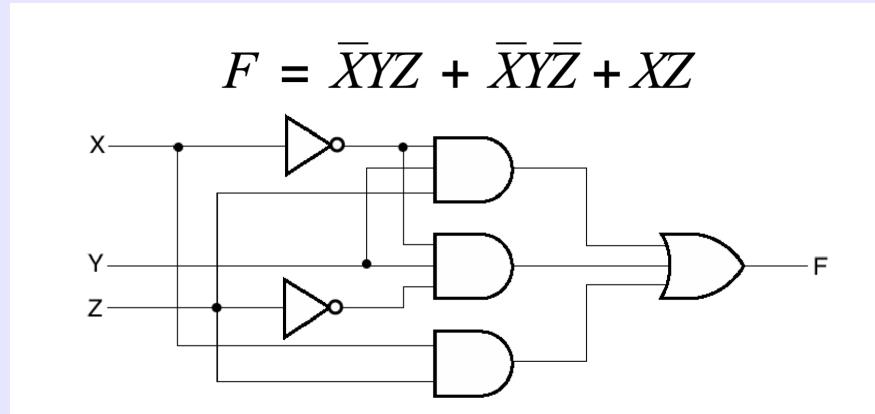


Figure 4: Here are the various ways wires can connect/not connect

Example

Here is an example of a circuit and the resulting algebra to "solve" it and how to simplify it



$$F = \bar{X}YZ + \bar{X}Y\bar{Z} + XZ$$

Apply 14. $X(Y+Z) = XY + XZ$

$$F = \bar{X}Y(Z + \bar{Z}) + XZ$$

Apply 7. $X + \bar{X} = 1$

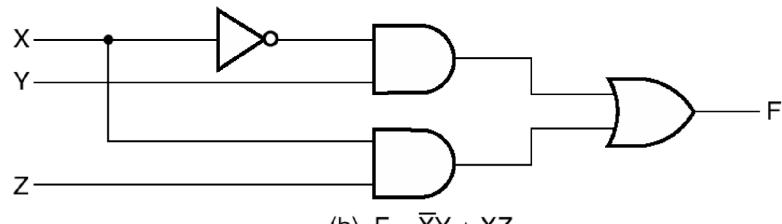
$$F = \bar{X}Y \cdot 1 + XZ$$

Apply 2. $X \cdot 1 = X$

$$F = \bar{X}Y + XZ$$

Fewer Gates

$$F = \bar{X}Y + XZ$$



Where output variables are either equivalent to 0 or 1 and input is the same. Moreover, simplifying this circuit and circuits in general allow for greater efficiency.

1.4 Standard Design Approach Sum of Products (SOP)

The three step approach:

1. Define truth table
2. Write down a Boolean expression for every row with the '1' in the output, for example, $Y = \overline{C}BA + \overline{C}BA + CBA + CBA$
3. Wire up all of the gates

Truth Table

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

Figure 5: Here is the truth table given the example

1.5 Karnaugh Map

Definition

Karnaugh maps, aka k-maps, are graphical representations of truth tables that use a grid with one cell for each row of the truth table

		B\A	00	01	11	10	
		C	0	0	1	1	0
		1	0	0	1	1	

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

Figure 6: An example k-map and its respective truth table!

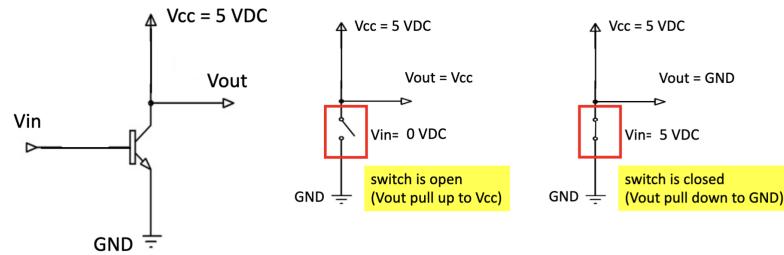
You pretty much put the 1's and 0's onto the cell given the values

Here are some rules given to the k-map

1. The grouping must be in the shape of a rectangle. There are no diagonal adjacencies allowed
2. All cells in the rectangle must contain ones. No zeros are allowed
3. The number of cells in groupings must be in powers of 2
4. Outside edges of K-maps are considered adjacent, so it may wrap around
5. Cells may be contained in more than one rectangle, but every rectangle must have at least ONE unique cell to
6. Every rectangle must be as large as possible
7. Everyone 1 must be covered by at least one rectangle

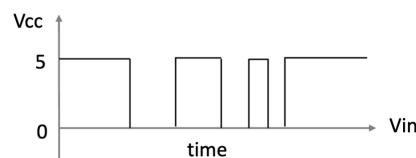
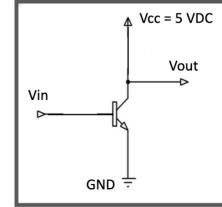
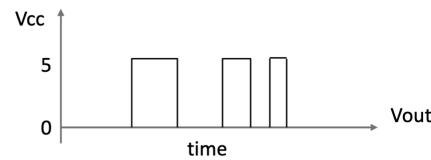
2 CMOS Gate Design and Analysis 1/27

The basic design of a transistor is as follows:



Basic operations:

Let's work through a simple example

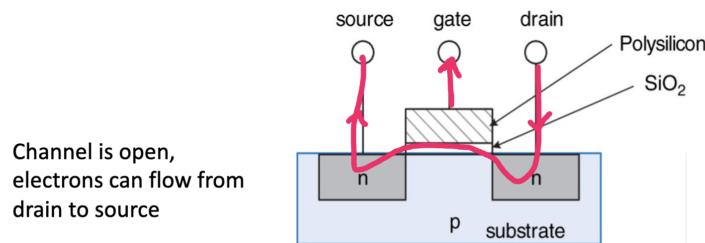
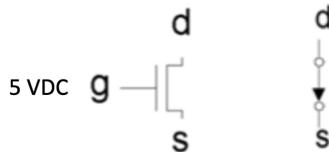


Input (V_{in}) voltage is "switching" the output (V_{out}) voltage

2.1 Metal Oxide Semiconductor (MOS) Transistor

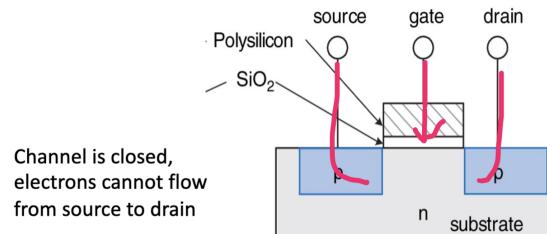
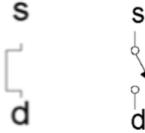
nMOS The n-channel Metal Oxide Semiconductor (nMOS) transistor

Switch is closed when
gate (g) has a positive
VDC value (e.g., 5 VDC).

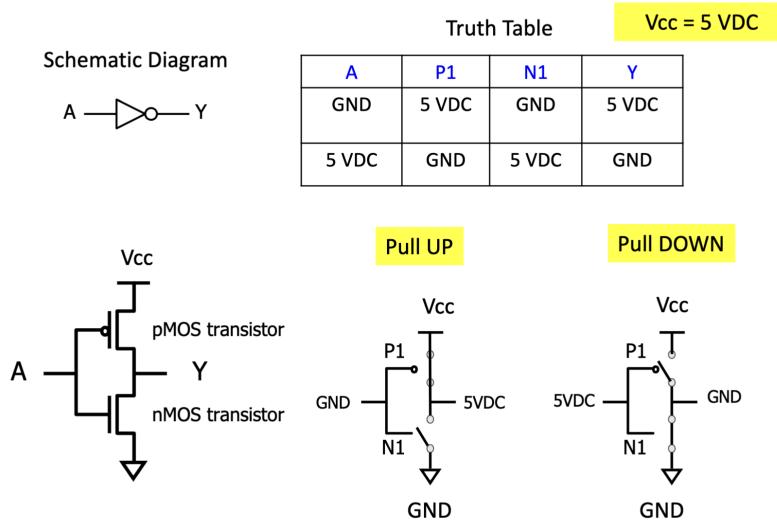


pMOS The p-channel Metal Oxide Semiconductor (pMOS) Transistor
It is similar to a dam, where the analogy states, there is a lot of "water" on one side and then directly flows down depending on amount of "water"

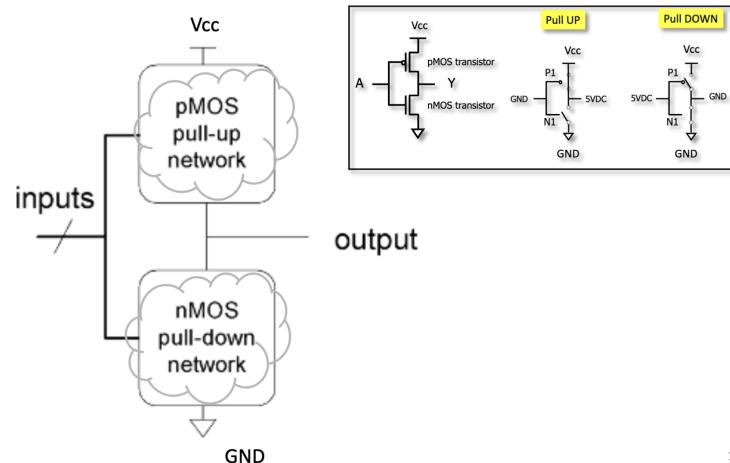
Switch is open when
gate has a positive VDC value (e.g., 5 VDC).



NOT Gate MOS Gate Design A strong 5V and strong no 5V



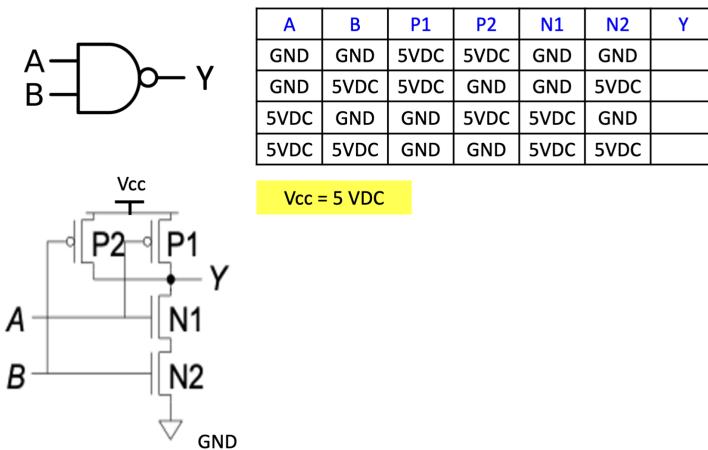
Complementary MOS Designs here



3 CMOS oscillator (clock) properties and design 1/29

This lecture, we are finishing up the remaining lecture from last time...

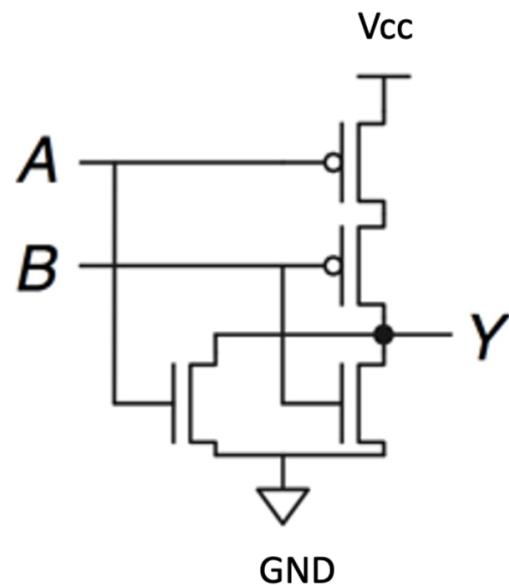
NAND Gate: MOS Design Only needs one or NONE of them on



15

For the first three, the output should be 5V, while the last output should be 0

NOR Gate: MOS Design Opposite of OR, where only 5V output when all off.



3.1 Relationship between Voltage and Logic

Digital Abstraction

Definition

Voltage a continuous value

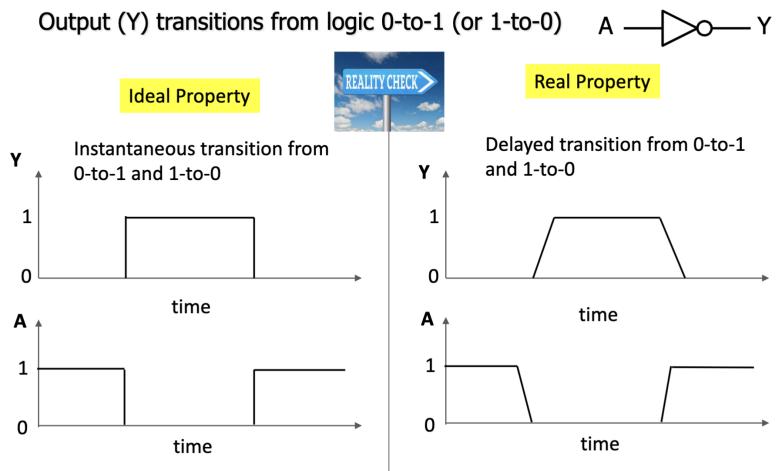
- Has a defined range of values, e.g. from 0 to 5VDC
- And any VDC value between, e.g., 0.1, 0.11, etc...
- Hardware understands voltage values

Boolean Logic (Logic) is a discrete value of 0 or 1

- Abstractions that humans understand
- Apply the rules of Boolean algebra
- Simplifies circuits

Continuous to Discrete conversion can be defined as having:

- Logic 1 - Has voltage range from 5 to 2 VDC
- Logic 0 - Has voltage range from 0 to 0.8 VDC
- Invalid - Less than 2 VDC, greater than 0.8 VDC, unreliable measurements

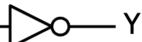


NOT Gate: Closer Inspection It will take time to transition from 0 to 1, and vice versa

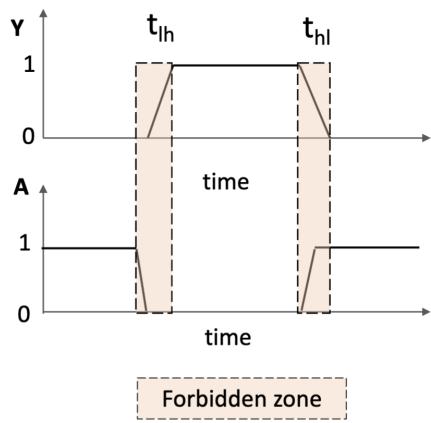
This moves onto to our definition of *Gate Delay*

Definition

Gate Delay is defined as the transition from logic 0-to-1 and vice versa

Output (Y) transitions from logic 0-to-1 (or 1-to-0) 

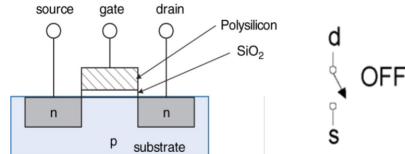
Real Property



t_{lh} = 0-to-1 (low to high) time delay
 t_{hl} = 1-to-0 (high to low) time delay

The amount of time (in seconds) needed for the output value to change (**propagation delay, t_d**)

In this course, we'll assume:
 $t_d = t_{lh} = t_{hl}$



Moving onto 1/29's actual lecture:

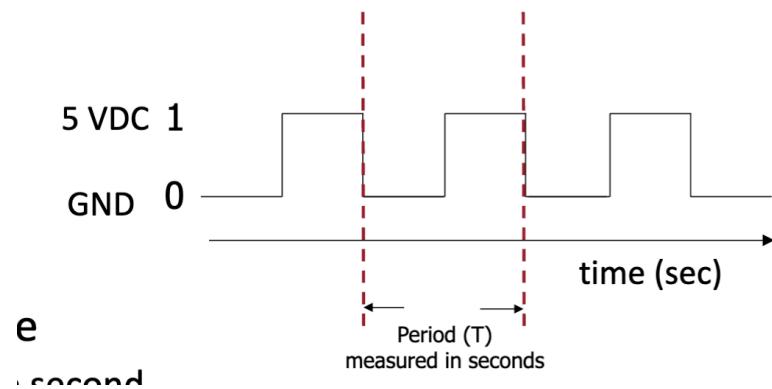
3.2 Clock and Clock design

Properties: Period and Frequency

- Clock period (T)
 - One **COMPLETE** cycle
 - Typical period of 1ns
 - Measured in **seconds**
- Clock Frequency (F) or rate
 - How many cycles in 1 seconds
 - Frequency = $F = \frac{1}{T}$

– Measured in **Hertz**

Shown below is what these ways would look like:



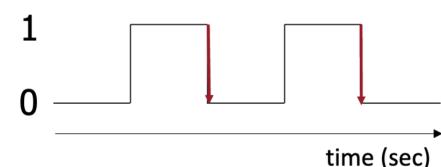
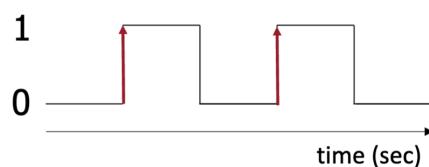
Properties: Events What are the different edges?

Rising-edge

- Signal transitions from logic 0 to logic 1
- What is the amount of time (sec) between two successive rising-edge events?

Falling-edge

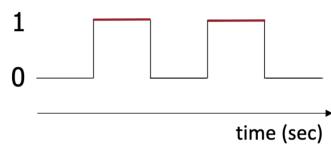
- Signal transitions from logic 1 to logic 0
- What is the amount of time (sec) between two successive falling-edge events?



Properties: Active High and Low Now lets look at the highs and lows

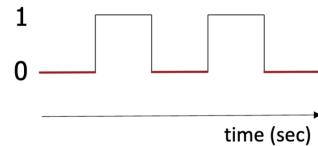
Active high

- Signal is logic 1
- What is the amount of time (sec) in one clock period?



Active low

- Signal is logic 0
- What is the amount of time (sec) in one clock period?



3.3 Clock design

Ring Oscillator clock that oscillates using inverter logic gates

Definition

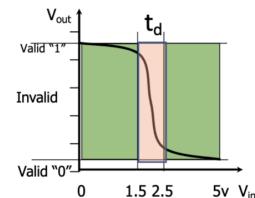
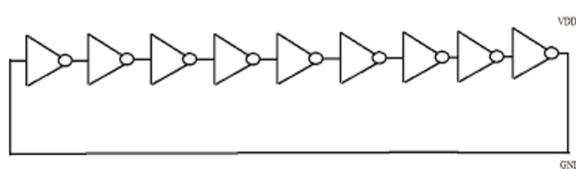
Clock Period is the propagation delay for a sequence of inverters

$$\text{Frequency} = 1 / (2 * \# \text{ of inverters} * t_d)$$

Can you think of a limitation?

Hint: # of inverters

- t_d = propagation delay for a single inverter
- propagation delay = amount of time (sec) for the output value to change when given an input value.

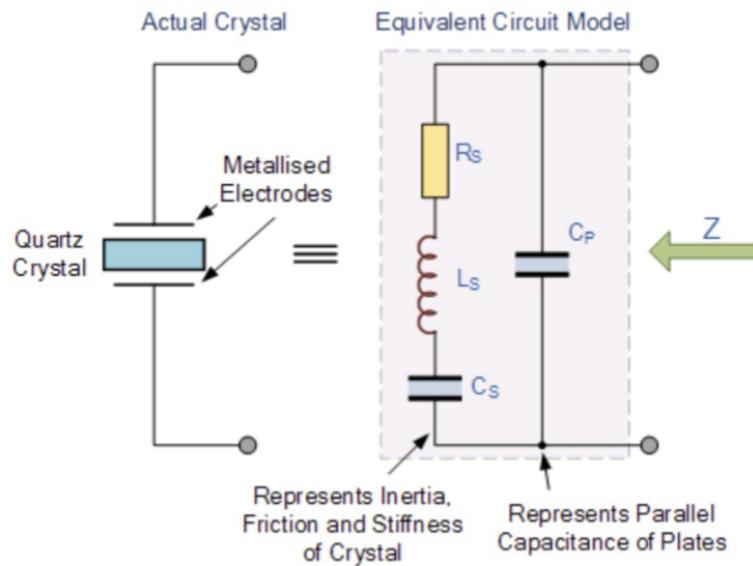


Quartz Crystal

Used in modern processor

Depending on the crystal's physical thickness and size, it can control:

- Frequency of oscillations
- Inversely proportional to its physical thickness between 2 metallic surfaces



4 Common components and analysis 2/3

There exists a Don't care (x)

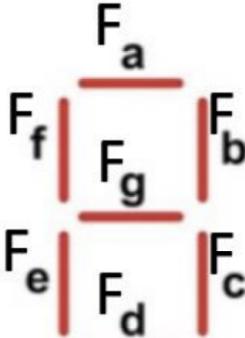
One example of this is our Lab 1

Example

Where if we don't need to display support letters A-F, we can put x ("don't care") in those rows

Where we can show it as:

7-Segment Display Layout



The diagram shows a 7-segment display layout with segments labeled F_a through F_g. The segments are arranged as follows: F_a is the top horizontal bar; F_b is the right vertical bar; F_c is the bottom right vertical bar; F_d is the bottom horizontal bar; F_e is the bottom left vertical bar; F_f is the left vertical bar; and F_g is the middle horizontal bar.

Inputs Outputs

A	B	C	D	F _a	F _b	F _c	F _d	F _e	F _f	F _g	
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0							
0	0	1	0	0	1						
0	0	1	1	0	1						
0	1	0	0	0	0						
0	1	0	1	0	1						
0	1	1	0	0	1						
0	1	1	1	0	1						
1	0	0	0	0	1						
1	0	0	1	0	1						
1	0	1	0	?	X	X	X	X	X	X	
1	0	1	1	?	X	X	X	X	X	X	
1	1	0	0	?	X	X	X	X	X	X	X
1	1	0	1	?	X	X	X	X	X	X	X
1	1	1	0	?	X	X	X	X	X	X	X
1	1	1	1	?	X	X	X	X	X	X	X

Don't care K-map rules:

- Because x denotes either 0 or 1, we treat it as a 0 or 1 in a K-map
- We circle an x if it helps us cover the 1's with larger or fewer rectangles, i.e. x is treated as 1
- We don't circle an x if it is not helpful for covering 1's, in this case we treat it as a 0

Example

K-map example 1

- Complete the K-map by drawing rectangle(s) that satisfy all K-map rules. Optionally, write the simplified Boolean expression.

AB\CD	00	01	11	10
00	0	0	x	1
01	0	1	x	1
11	0	1	x	x
10	0	0	x	x

$$\textcolor{red}{C} + \textcolor{green}{BD}$$

K-map example 2

- Complete the K-map by drawing rectangle(s) that satisfy all K-map rules. Optionally, write the simplified Boolean expression.

AB\CD	00	01	11	10
00	1	1	x	1
01	1	0	x	1
11	1	1	x	x
10	1	0	x	x

$$\textcolor{red}{D} + \textcolor{green}{A}\bar{B} + \textcolor{blue}{AB}$$

4.1 Multiplexer (MUX)

S bits to "select" which input (A or B) becomes the output (Y)

$$S = 1 \rightarrow Y = B$$

$$S = 0 \rightarrow Y = A$$

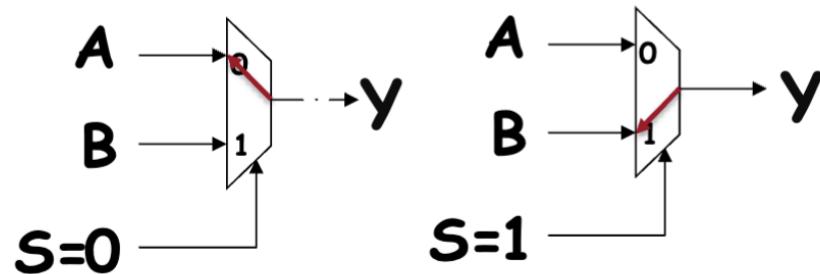


Figure 7: Example of the multiplexer

Number of inputs = 2 to the power of whatever number of select bits
 If we have 4 inputs, A, B, C, D, then we would need 2 signal bits for example

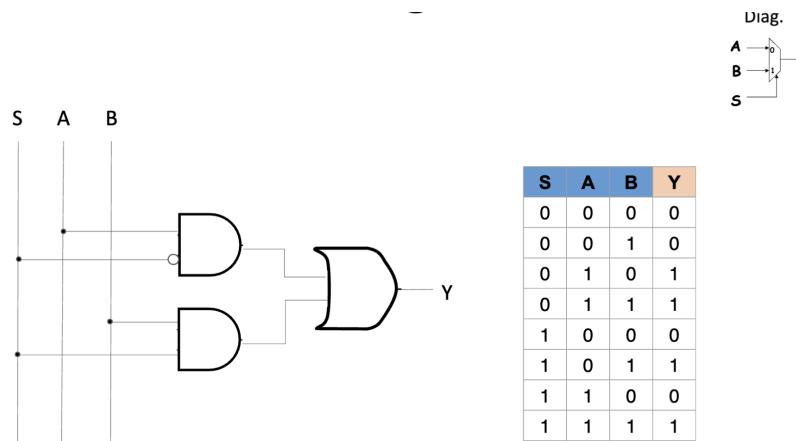


Figure 8: Truth table and circuit with two inputs of MUX

4.2 Demultiplexer (DeMUX)

S bits to "select" which input (A or B) becomes the output (Y)

$$S = 1 \rightarrow A = Y$$

$$S = 0 \rightarrow B = Y$$

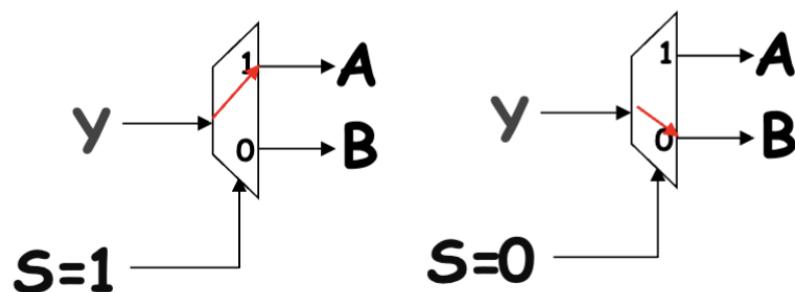


Figure 9: Example of the Demultiplexer

Number select bits = $\log_2(\text{number of output bits})$

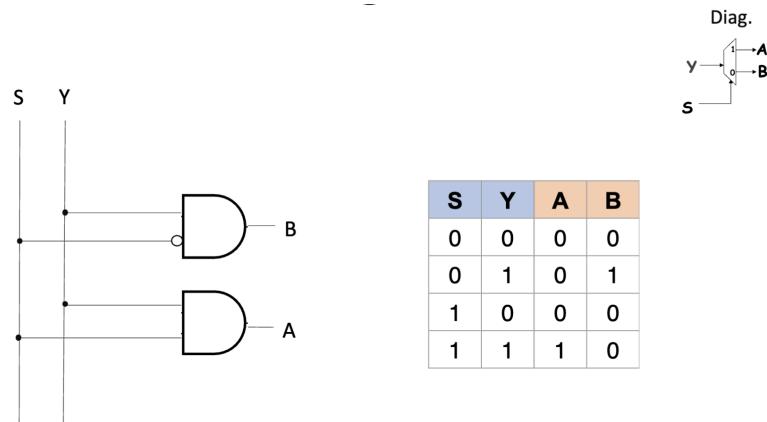


Figure 10: Truth table and circuit for DeMUX

4.3 Decoder

S input bits are used to "select" which output bits (A) are turned on (logic 1)

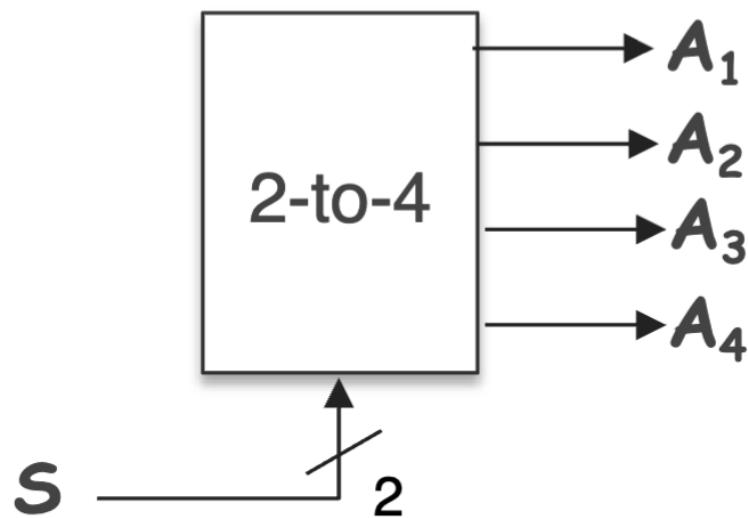


Figure 11: Decoder example

The number of outputs = 2 to the power of number of select bits

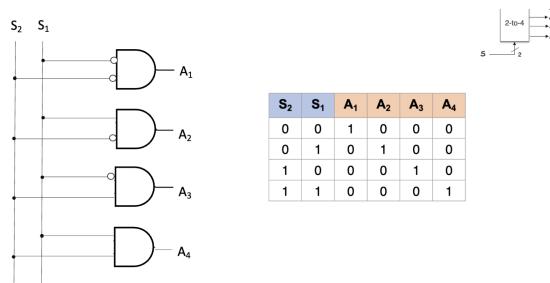


Figure 12: Truth table and circuit for Decoder

4.4 Encoder

Input bit (A) are used to select which output bits (S) are turned on (logic 1)

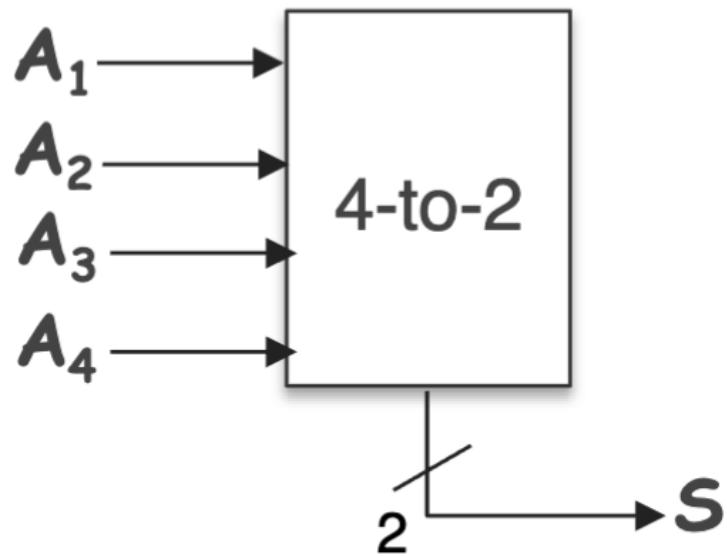


Figure 13: Encoder example

$$\text{Number of output bits} = \log_2(\text{number of input bits})$$

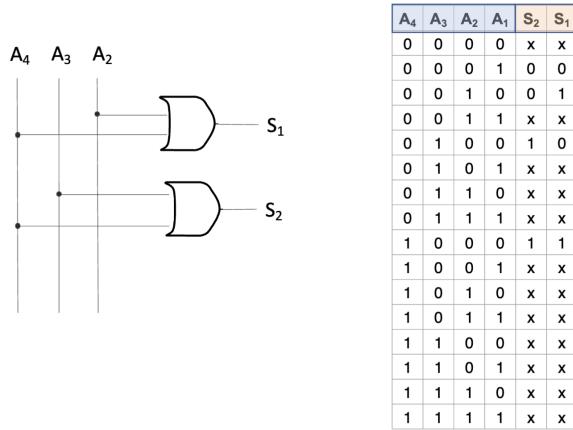


Figure 14: Truth table and circuit for Encoder

5 ALU component design and analysis 2/5

What does the ALU do? It allows computer to add, divide, etc. also allows for bit shifts and what not

5.1 Add and Subtract

Binary Addition not a single operation

Where $A + B = \text{sum}$ and carry-out
 $A, B, \text{Sum } (S)$ and Carry-out (C_0) are one bit binary values as seen below:

Four possibilities (A and B):

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

Binary Half Adder Circuit Design where there are two input bits, A and B holding 1 bit each. And two output bits, C and S, holding 1 bit each

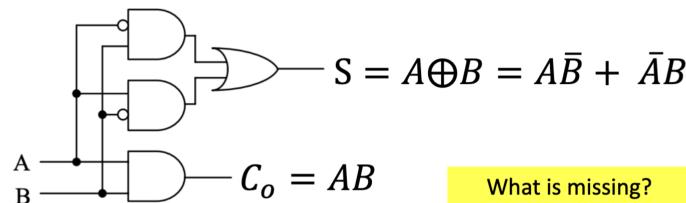


Figure 15: Half Adder Circuit

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 16: Truth table for half adder circuit

Binary Full Adder Circuit Design Similar to the half adder, where we have three input bits: C_{in} , A, B, where they have 1 bit each

And two output bits: C_0 and S, that have 1 bit each.

S: 

bit each

S: 

bit each

C _i	A	B	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$C_o = C_i(A + B) + AB = C_i(A \oplus B) + AB$$

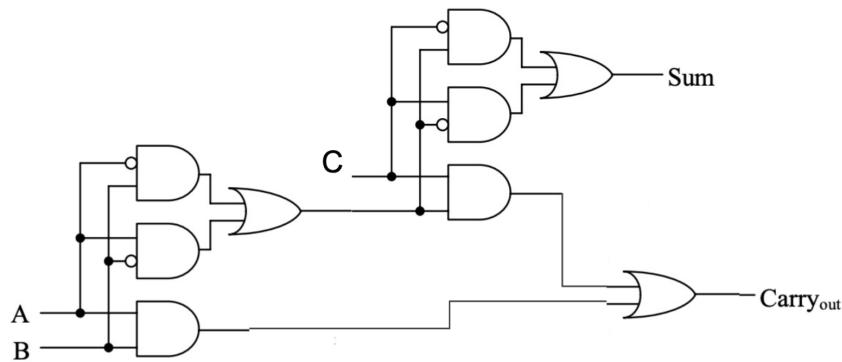
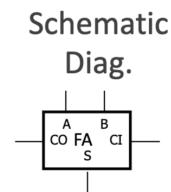
$$S = C_i \oplus A \oplus B$$

Figure 17: The truth table and equations for the Binary Full Adder

Moreover, we can take a look at the full adder circuit itself:

Full Adder Circuit

Two HA circuits plus one or gate



5.2 FA Component Properties

We can assume that the propagation delay (t_d) for each logic gate is 1 nanosecond (ns).

Sum bit analysis We can get the value of the output (S) and we can also get the delay, t_d

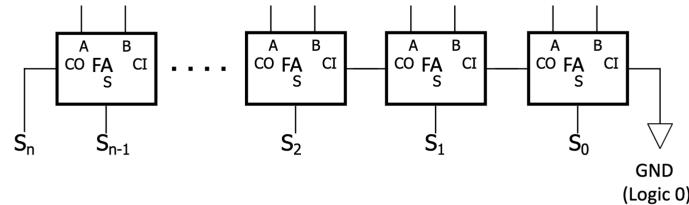
Therefore, we can also get worst case analysis, where it is the max(Sum t_d , Carry-out t_d)

In the case of the full adder, we know that the sum is 2ns for worst case, and carry-out is 3ns for worst case.

Thus, all component outputs will be stable in 3ns

5.3 Add and subtract Circuit

Using FA circuits, extend to arbitrary number of bits



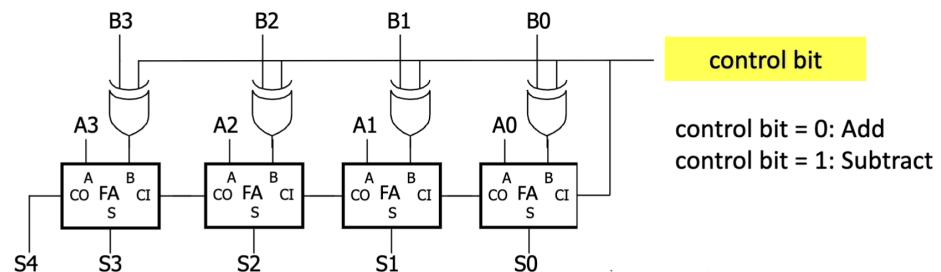
"Ripple-Carry Adder"

- carries ripple through from right to left
- longest chain of carries has length n

Subtract A-B: 2's complement Operation

$\sim = \text{bit-wise complement}$

2's complement: $-B = \sim B + 1$

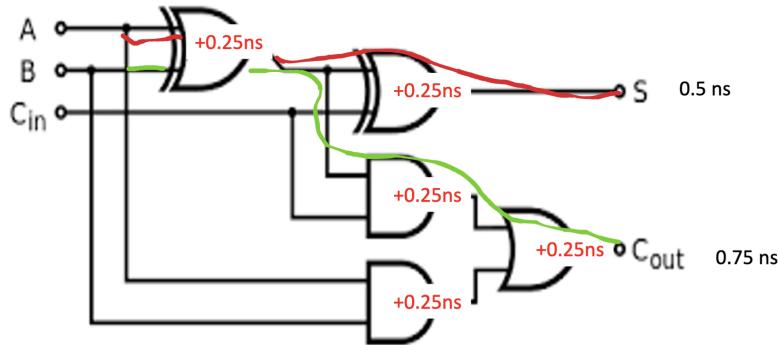


Carry-out bit We'll see how this is used very soon!

Figure 18: Here is a 4 bit example of addition and subtract

Looking back at the Full Adder, we see that the worst case per component is this:

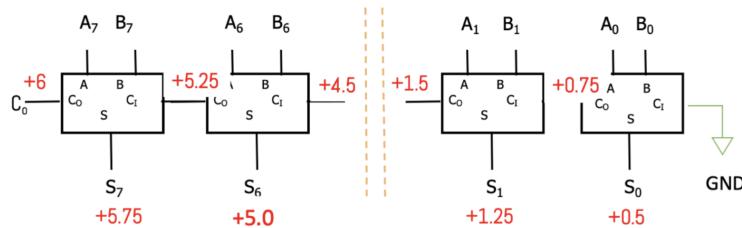
Worst case analysis (i.e., upper bound)



Therefore, we when we take a look at the 8-bit full add circuit:

If the clock period is 1 ns, then how many clock cycles (worst case) are needed?

- Carry out delay (t_d) = 0.75 ns
- Total delay = $t_d \times \text{number of bits} = 0.75 \times 8 = 6 \text{ ns}$ (**6 clock cycles are needed**).

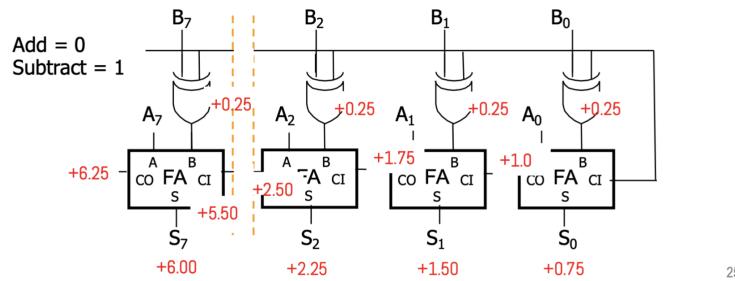


As a side note, the reason why S_1 is 1.5 ns is due to the fact that S and C_{out} is different in structure and has different delays

Full Add and Subtract Circuit Here we have a 8-bit full add and subtract circuit that is actually in parallel.

If the clock period is 1 ns, then how many clock cycles (worst case) are needed?

- LSB (C_0) carry-out delay (t_{d0}^0) = 1.0 ns
- FA carry-out delay (t_d) = 0.75 ns
- Total delay = $t_{d0}^0 + t_d \times (\text{number of bits} - 1) = 1.0 + 0.75 \times 7 = 6.25 \text{ ns}$
(7 clock cycles are needed)



25

5.4 Bit Shift Circuit

Recall

Here is a **reminder** of what bit shifts are!

Left Shift: shifts in a 0 from the right end

- $(X << 1) = 00101000_2 = 40_{10}$

$$X = 20_{10} = 00010100_2$$

"Logic" Right Shift: shifts in a 0 from the left end

- $(X >> 1) = 00001010_2 = 10_{10}$

"Arithmetic" Right Shift: maintains the sign bit

- $-X = -20_{10} = 2\text{'s complement of } X = 11101100_2$
- $(-20_{10} >>> 1) = (11101100_2 >>> 1) = 11110110_2 = -10_{10}$

Note:

- shift right arithmetic notation ($>>>$)
- shift right logic notation ($>>$)

Bit Shift Component Design But how do we do this with a circuit? We can do it here!

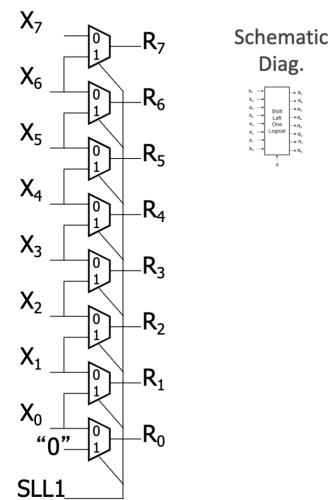
Example Shift Left Circuit

If SLL1 is true (logic 1)

- Shifts the input X one bit to the left
- $R \leftarrow X \ll 1$

If SLL1 is false (logic 0)

- Do not shift X
- $R \leftarrow X$



Shift left by 2

- Rewire the multiplexors so each X_i feeds into R_{i+2}
- Similarly: shift left by 4, etc.

Shift right circuits have similar circuitry

- shift right logical: each X_i feeds into a lower numbered R_j
- shift right arithmetic: sign bit stays the same

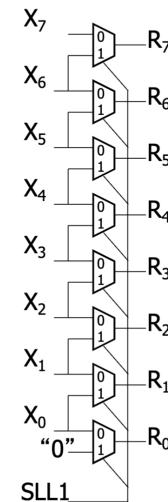


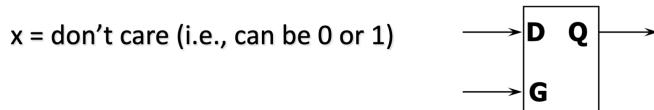
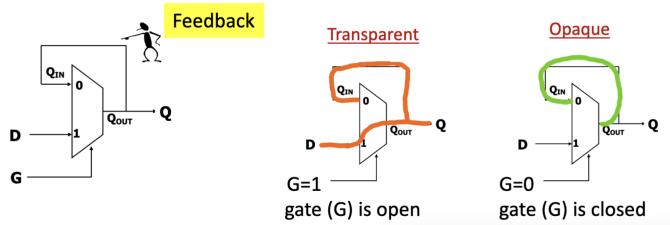
Figure 19: Here is an example of the bitshift with this circuit

6 Latches, D Flip-Flops (introduce J/K and T Flip-Flops), and memory/register design 2/24

6.1 Latch Component

stores 1 bit

- Store 1 bit (0 or 1)
- Storage = stateful = has memory!
- 2-to-1 MUX with Feedback
- Inputs D, Q_{in} (or Q), G (switch or gate bit)
- Output Q_{out}
- Q has both an input and output



G	D	Q_{IN}	Q_{OUT}
opaque	[0 X 0]	[0 X 1]	[0 1]
transparent	[1 0 X]	[1 1 X]	[0 1]

stable
unstable

Figure 20: Truth table for Latch

Timing Waveform for a Latch Amount of time before stable

