EECS150 - Digital Design
Lecture 2 - Synchronous Digital Systems Review Part 1

January 21, 2010

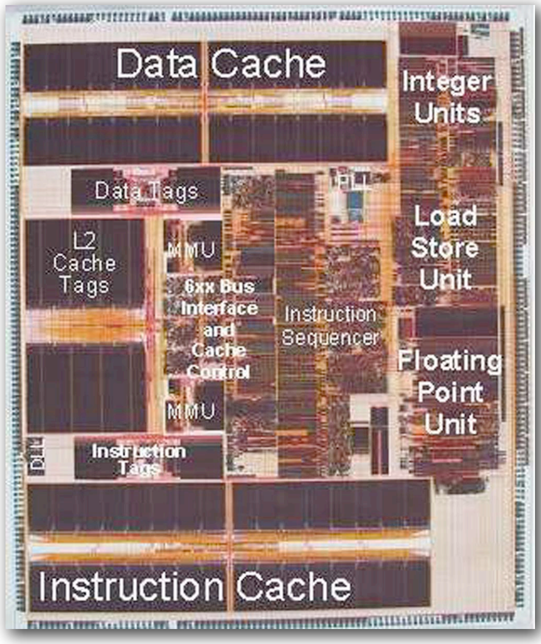
John Wawrzynek
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www-inst.eecs.berkeley.edu/~cs150

Outline

- Topics in the review, you have already seen in CS61C, and possibly EE40:
  1. Digital Signals.
  2. General model for synchronous systems.
  3. Combinational logic circuits
  4. Flip-flops, clocking (Next week)

Integrated Circuit Example

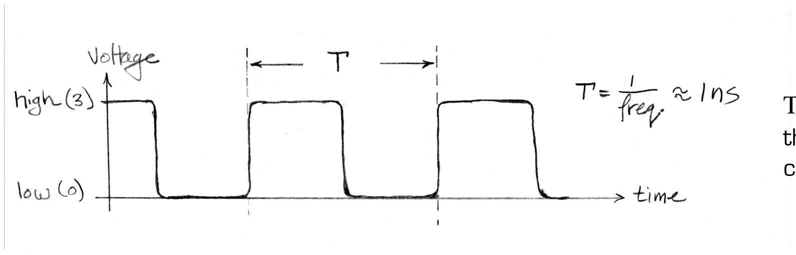


- PowerPC microprocessor micro-photograph
  - Superscalar (3 instructions/cycle)
  - 6 execution units (2 integer and 1 double precision IEEE floating point)
  - 32 KByte Instruction and Data L1 caches
  - Dual Memory Management Units (MMU)
  - External L2 Cache interface with integrated controller and cache tags.

Comprises only transistors and wires.

Connections to outside world (ex. motherboard)
- Memory interface
- Power (Vdd, GND)
- Clock input

Clock Signal



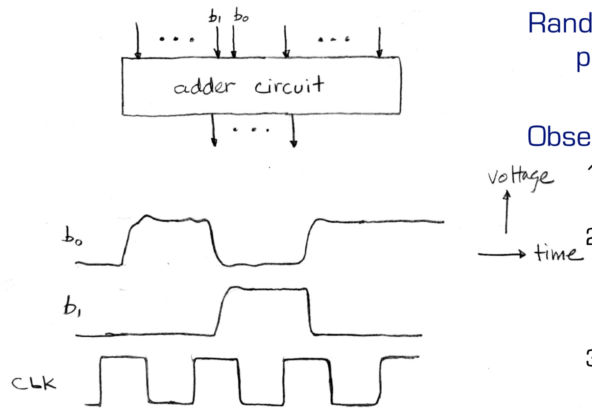
T represents the time of one clock "cycle".

A source of regularly occurring pulses used to measure the passage of time.

- Waveform diagram shows evolution of signal value (in voltage) over time.

- Usually comes from an off-chip crystal-controlled oscillator.
- One main clock per chip/system.
- Distributed throughout the chip/system.
- "Heartbeat" of the system. Controls the rate of computation by directly controlling all data transfers.

Data Signals



Random adder circuit at a random point in time:

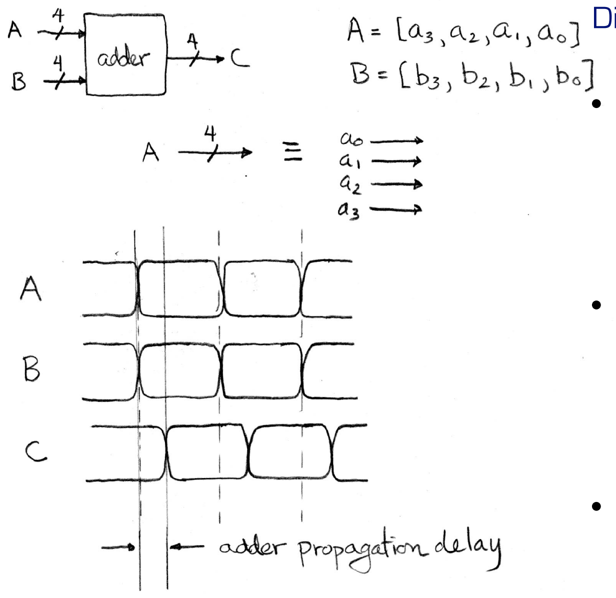
Observations:

1. Most of the time, signals are in either low- or high-voltage position.
2. When the signals are at the high- or low-voltage positions, they are not all the way to the voltage extremes (or they are past).
3. Changes in the signals correspond to changes in clock signal (but don't change every cycle).

The facts:

1. Low-voltage represents binary 0 and high-voltage, binary 1.
2. Circuits are design and built to be "restoring". Deviations from ideal voltages are ignored. Outputs close to ideal.
3. In synchronous systems, all changes follow clock edges.

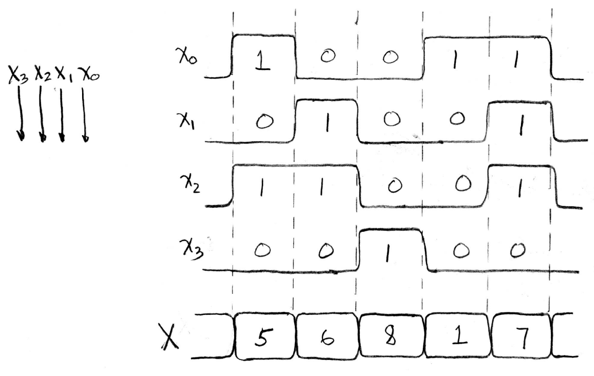
Circuit Delay



Digital circuits cannot produce outputs instantaneously.

- In general, the delay through a circuit is called the propagation delay. It measures the time from when inputs arrive until the outputs change.
- The delay amount is a function of many things. Some out of the control of the circuit designer:
  - Processing technology, the particular input values.
- And others under her control:
  - Circuit structure, physical layout parameters.

Bus Signals

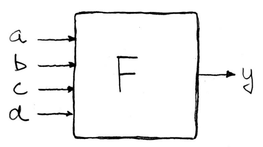


Signal wires grouped together often called a bus.

- X_0 is called the least significant bit (LSB)
- X_3 is called the most significant bit (MSB)
- Capital X represents the entire bus.
  - Here, hexadecimal digits are used to represent the values of all four wires.
  - The waveform for the bus depicts it as being simultaneiously high and low. (The hex digits give the bit values). The waveform just shows the timing.

Combinational Logic Blocks

- Example four-input function:

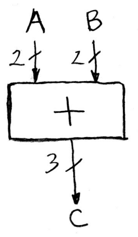


- True-table representation of function. Output is explicitly specified for each input combination.
- In general, CL blocks have more than one output signal, in which case, the truth-table will have multiple output columns.

a b c d	y
0 0 0 0	F(0,0,0,0)
0 0 0 1	F(0,0,0,1)
0 0 1 0	F(0,0,1,0)
0 0 1 1	F(0,0,1,1)
0 1 0 0	F(0,1,0,0)
0 1 0 1	F(0,1,0,1)
0 1 1 0	F(0,1,1,0)
1 1 1 1	F(0,1,1,1)
1 0 0 0	F(1,0,0,0)
1 0 0 1	F(1,0,0,1)
1 0 1 0	F(1,0,1,0)
1 0 1 1	F(1,0,1,1)
1 1 0 0	F(1,1,0,0)
1 1 0 1	F(1,1,0,1)
1 1 1 0	F(1,1,1,0)
1 1 1 1	F(1,1,1,1)

Example CL Block

- 2-bit adder. Takes two 2-bit integers and produces 3-bit result.



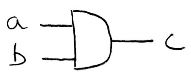
- Think about true table for 32-bit adder. It's possible to write out, but it might take a while!

a1 a0	b1 b0	c2 c1 c0
0 0	0 0	0 0 0
0 0	0 1	0 0 1
0 0	1 0	0 1 0
0 0	1 1	0 1 1
0 1	0 0	0 0 1
0 1	0 1	0 1 0
0 1	1 0	0 1 1
0 1	1 1	1 0 0
1 0	0 0	0 1 0
1 0	0 1	0 1 1
1 0	1 0	1 0 0
1 0	1 1	1 0 1
1 1	0 0	0 1 1
1 1	0 1	1 0 0
1 1	1 0	1 0 1
1 1	1 1	1 1 0

Theorem: *Any* combinational logic function can be implemented as a networks of logic gates.

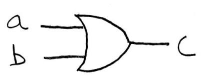
Logic "Gates"

AND



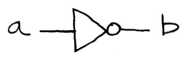
ab	c
00	0
01	0
10	0
11	1

OR



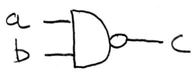
ab	c
00	0
01	1
10	1
11	1

NOT



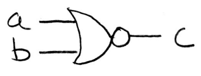
a	b
0	1
1	0

NAND



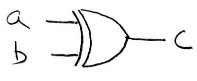
ab	c
00	1
01	1
10	1
11	0

NOR



ab	c
00	1
01	0
10	0
11	0

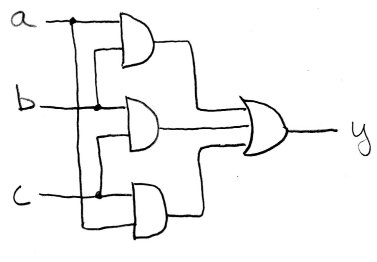
XOR



ab	c
00	0
01	1
10	1
11	0

- Logic gates are often the primitive elements out of which combinational logic circuits are constructed.
  - In some technologies, there is a one-to-one correspondence between logic gate representations and actual circuits.
  - Other times, we use them just as another abstraction layer (FPGAs have no real logic gates).
- How about these gates with more than 2 inputs?
- Do we need all these types?

Example Logic Circuit

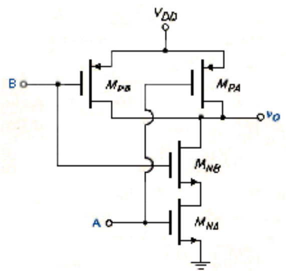


a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

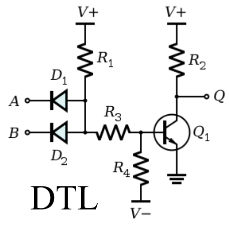
- How do we know that these two representations are equivalent?

Logic Gate Implementation

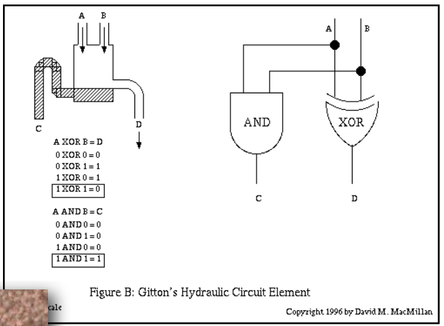
- Logic circuits have been built out of many different technologies. As we know, as long as we have a basic logic gate (AND or OR) and inversion we can build any a complete logic family.



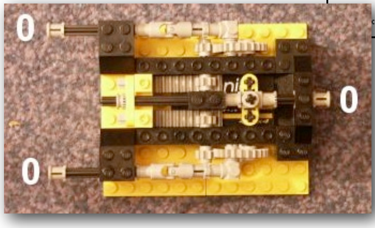
CMOS Gate



DTL



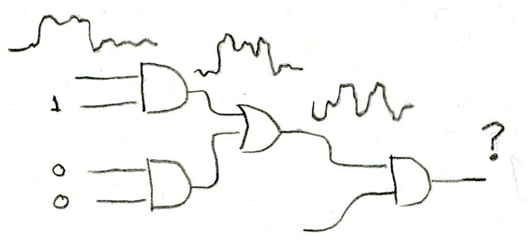
Hydraulic



Mechanical LEGO logic gates. A clockwise rotation represents a binary “one” while a counter-clockwise rotation represents a binary “zero.”

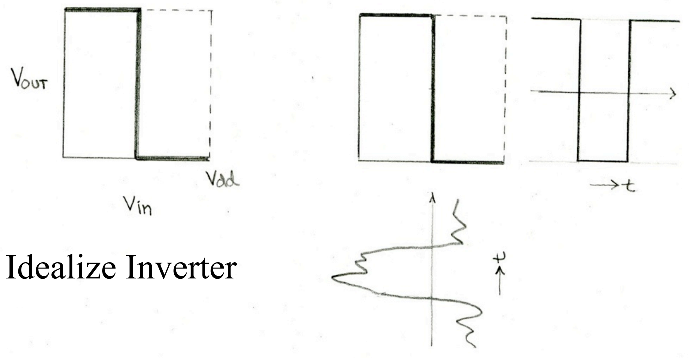
Restoration

- An necessary property of any successful technology for logic circuits is "Restoration".
- Circuits need:
  - to ignore noise and other non-idealities at the their inputs, and
  - generate "cleaned-up" signals at their output.
- Otherwise, each stage would propagates input noise to their output and eventually noise and other non-idealities would accumulate and signal content would be lost.

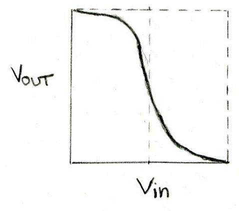


Inverter Example of Restoration

Example (look at 1-input gate, to keep it simple):



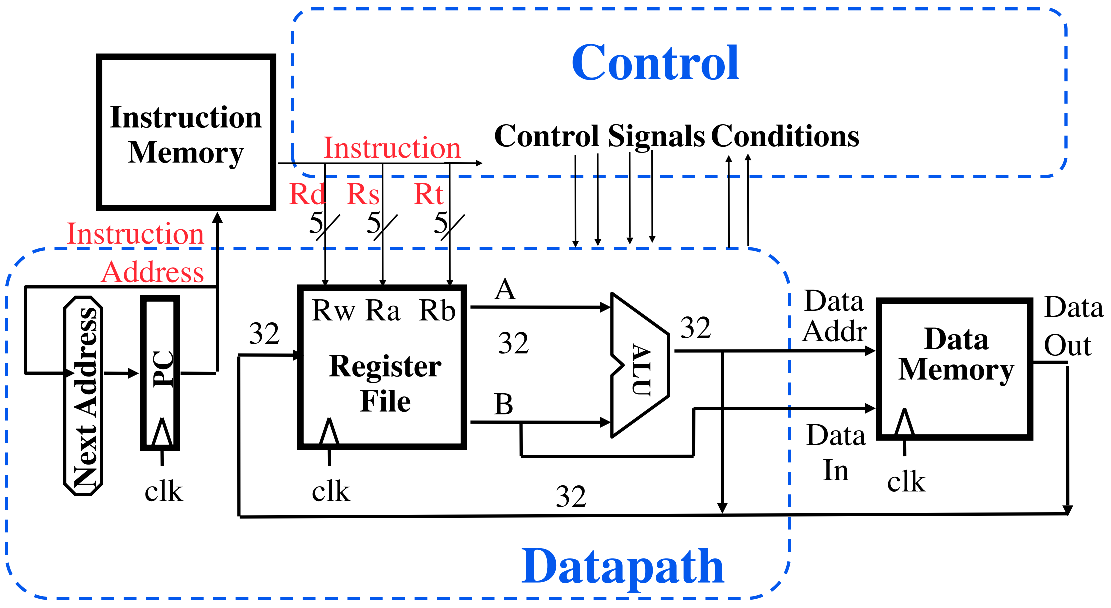
Idealize Inverter



Actual Inverter

- Inverter acts like a "non-linear" amplifier
- The non-linearity is critical to restoration
- Other logic gates act similarly with respect to input/output relationship.

Abstract View of MIPS Implementation



How do we implement these various pieces?

MIPS ALU Functions

- Responsible for the action taken by most of the R-type instructions: add, sub, and, or, …
- Arithmetic operations are complex. We'll study those later (although in 61c you saw a simple "ripple adder/subtractor")
- "Bitwise logical" instructions (and, or, …) take values from 2 registers and combine them according to some logic operation.
- Example: `and $r3, $r2, $r1`
- Implementation within the ALU:

- Likewise for or, exor, …

MIPS Implemenation

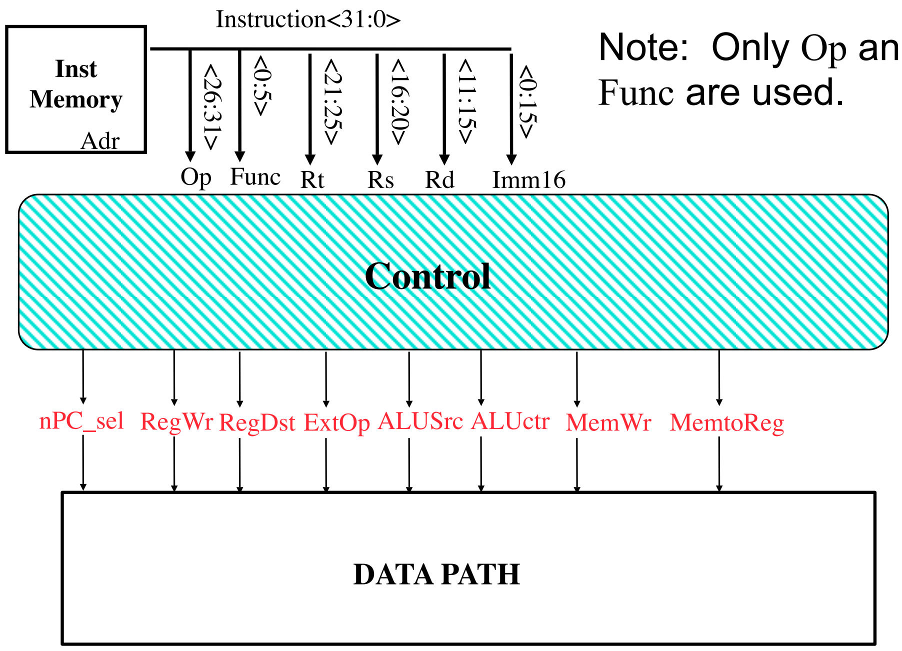
- Consider `beq` instruction:

```
        beq  $2,$1,loop
```

- How does the processor check to see if the two register values are equal?
- One approach (used in 61c) is to subtract the two values and check the result for zero (all bits of the result are 0).
- Okay, so how does the processor check the result for zero?

- What if the we can't use the subtractor to compare the two register values. Is it possible to compare them directly?

61c MIPS, a Combinational Logic Block



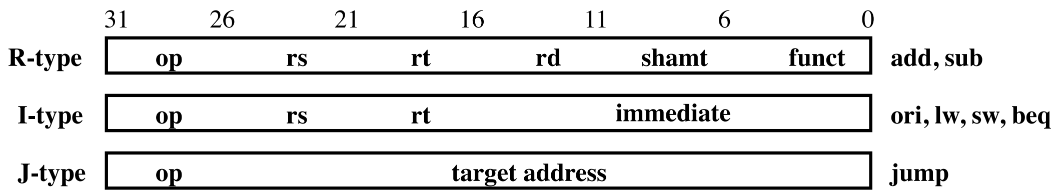
Note: Only Op and Func are used.

MIPS Controller Implementation

- The controller examines the instruction as it comes from the instruction memory (or cache), “decodes” it, and asserts the proper “control signals” to be used by the rest of the processor for instruction execution.
- Instruction decoding is the process of identifying the instruction type and operation code.
- Then based on the instruction operation code, the proper control signals can be asserted.

61C MIPS Controller Summary

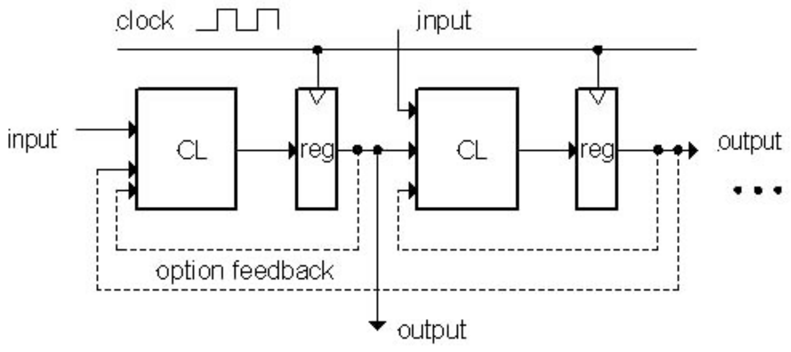
func	10 0000	10 0010	**We Don’t Care :-)**				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	**sub**	**ori**	**lw**	**sw**	**beq**	**j**
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<1:0>	Add	Subtract	Or	Add	Add	Subtract	xxx



Instruction Decoding

- Jump instruction. Op = 000010
- Branch if equal instruction. Op = 000100
- Store word instruction. Op = 101011
- The instruction decode would assert a special signal for each of these instructions:

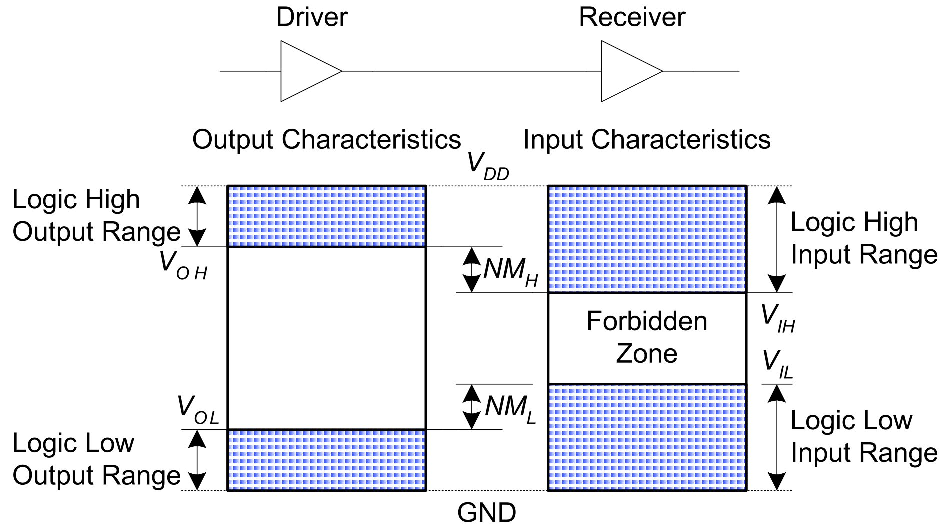
General Model for Synchronous Systems



- All synchronous digital systems fit this model:
  - Collections of combinational logic blocks and state elements connected by signal wires. These form a directed graph with only two types of nodes (although the graph need not be bi-partite.)
  - Instead of simple registers, sometimes the state elements are large memory blocks.

Extras

Noise Margins

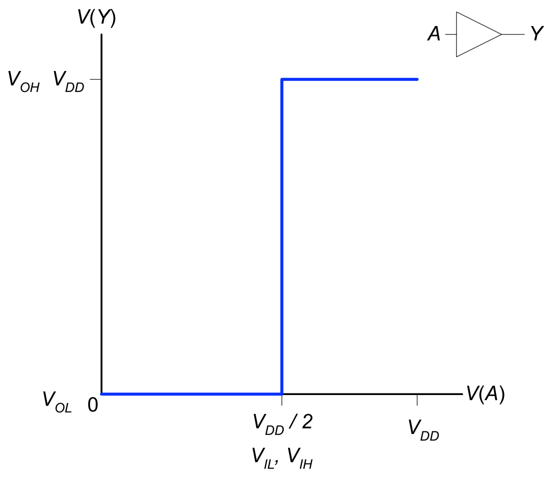


$$NM_H = V_{OH} - V_{IH}$$

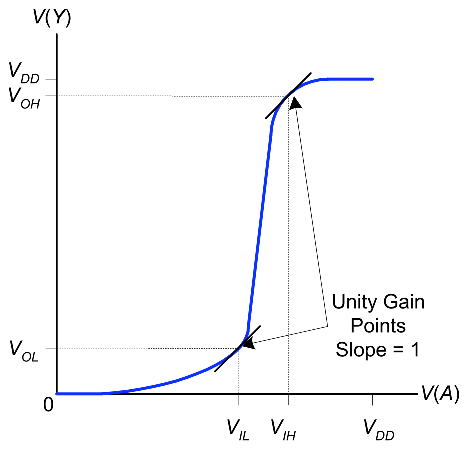
$$NM_L = V_{IL} - V_{OL}$$

D.C. Transfer Characteristics

Ideal Buffer:



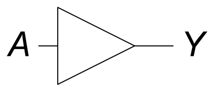
Real Buffer:

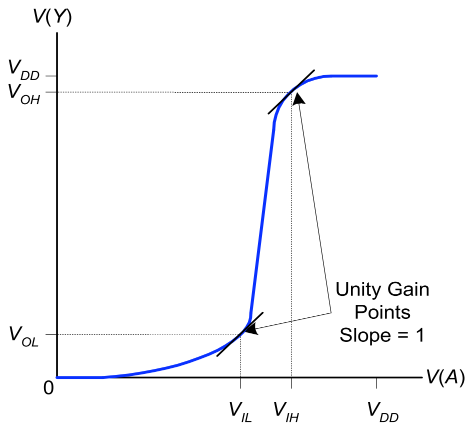


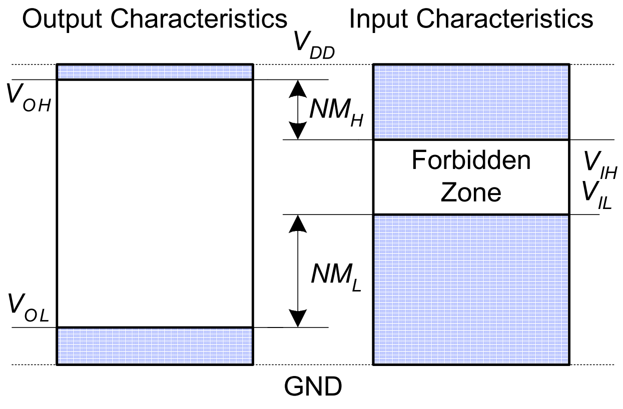
$NM_H = NM_L = V_{DD}/2$

$NM_H, NM_L < V_{DD}/2$

D.C. Transfer Characteristics







V_{DD} Scaling

- Chips in the 1970's and 1980's were designed using $V_{DD} = 5$ V
- As technology improved, V_{DD} dropped
  - Avoid frying tiny transistors
  - Save power
- 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V, 1.0 V, …
- Be careful connecting chips with different supply voltages

Chips operate because they contain magic smoke

Proof:

- if the magic smoke is let out, the chip stops working