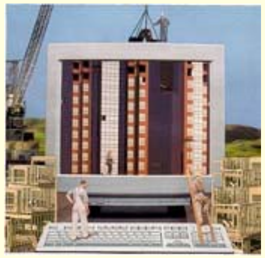


Logic Manipulation

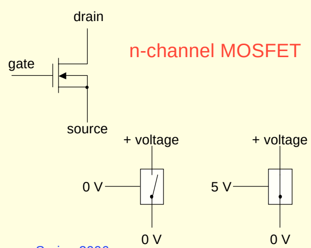
Transistors
Digital Logic
Computers

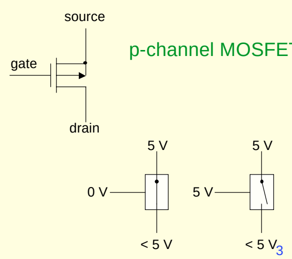
What does a computer do?

- Computers *store* and *manipulate* information
- Information is represented digitally, as voltages
- Digital format avoids ambiguity
  - below 1.5 V interpreted as 0 (5V CMOS logic)
  - above 3.5 V interpreted as 1 (5V CMOS logic)
- Information can be manipulated in many ways:
  - can be compared to other information
  - mathematical operations
  - define state of devices (display, speakers, motors, etc.)

Transistors: The Main Building Block

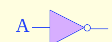
- Transistors, as applied to logic designs, act as voltage-controlled switches
  - n-channel MOSFET is closed when positive voltage (+5 V) is applied, open when zero voltage
  - p-channel MOSFET is open when positive voltage (+5 V) is applied, closed when zero voltage
    - (MOSFET means metal-oxide semiconductor field effect transistor)





Data manipulation

- All data manipulation is based on *logic*
- Logic follows well defined rules, producing predictable digital output from certain input
- Examples:



NOT

A	C
0	1
1	0

AND

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

OR

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

XOR

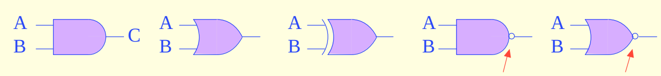
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

NAND

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

NOR

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

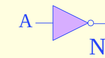


bubbles mean inverted (e.g., NOT AND → NAND)

Logic Gates

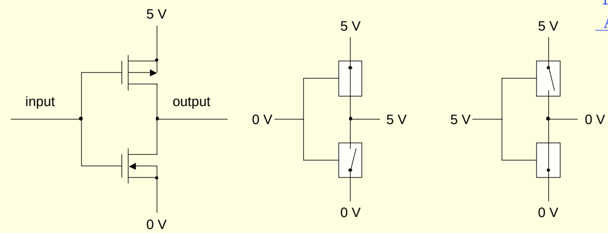
- The logic operations are carried out electronically by *gates*, represented by the symbols just introduced
- Gates are constructed out of *transistors*, typically 4–6 per gate
- Transistors simply act like *switches*, controlling data flow
- Gate response is typically ~1 nanosecond (1 billionth sec.)
- Can theoretically build an entire computer using only NAND (or NOR) gates…
  - And then you can take over the world! (sinister laugh…)

An inverter (NOT) from MOSFETS:



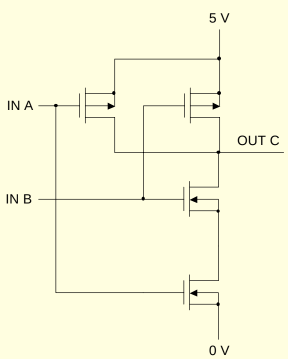
NOT

A	C
0	1
1	0



- 0 V input turns OFF lower (n-channel) FET, turns ON upper (p-channel), so output is connected to +5 V
- 5 V input turns ON lower (n-channel) FET, turns OFF upper (p-channel), so output is connected to 0 V
  - Net effect is logic inversion: 0 → 5; 5 → 0
- Complementary MOSFET pairs → CMOS

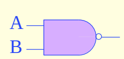
A NAND gate from scratch:



- Both inputs at zero:
  - lower two FETs OFF, upper two ON
  - result is output HI
- Both inputs at 5 V:
  - lower two FETs ON, upper two OFF
  - result is output LOW
- IN A at 5V, IN B at 0 V:
  - upper left OFF, lowest ON
  - upper right ON, middle OFF
  - result is output HI
- IN A at 0 V, IN B at 5 V:
  - opposite of previous entry
  - result is output HI

NAND

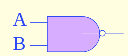
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0



NAND-based gate construction

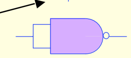
NAND

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



NOT

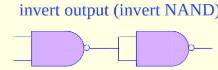
A	C
0	1
1	0



AND

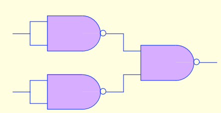
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

invert output (invert NAND)



OR

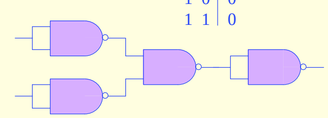
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1



invert both inputs

NOR

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



invert inputs *and* output (invert OR)

Arithmetic Example

- Let's add two binary numbers:

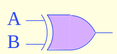
  00101110 = 46
  + 01001101 = 77
  01111011 = 123

- How did we do this? We have rules:

  0 + 0 = 0; 0 + 1 = 1 + 0 = 1; 1 + 1 = 10 (2): (0, carry 1);
  1 + 1 + (carried 1) = 11 (3): (1, carry 1)

- Rules can be represented by gates
  - If two input digits are A & B, output digit looks like XOR operation (but need to account for carry operation)

XOR



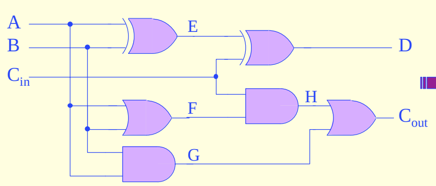
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

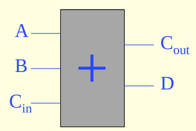
Can make rule table:

C_{in}	A	B	D	C_{out}
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

- Digits A & B are added, possibly accompanied by carry instruction from previous stage
- Output is new digit, D, along with carry value
  - D looks like XOR of A & B when C_{in} is 0
  - D looks like XNOR of A & B when C_{in} is 1
  - C_{out} is 1 if two or more of A, B, C_{in} are 1

Binary Arithmetic in Gates





"Integrated" Chip

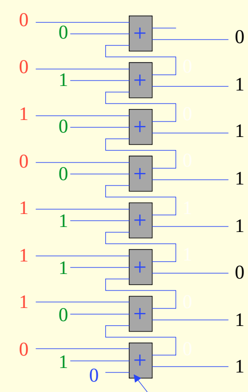
Input			Intermediate				Output	
A	B	C_{in}	E	F	H	G	D	C_{out}
0	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0
1	0	0	1	1	0	0	1	0
1	1	0	0	1	0	1	0	1
0	0	1	0	0	0	0	1	0
0	1	1	1	1	1	0	0	1
1	0	1	1	1	1	0	0	1
1	1	1	0	1	1	1	1	1

Each digit requires 6 gates

Each gate has ~6 transistors

~36 transistors per digit

8-bit binary arithmetic (cascaded)



MSB

00101110 = 46
+ 01001101 = 77
01111011 = 123

Carry-out tied to carry-in of next digit.

"Magically" adds two binary numbers

Up to ~300 transistors for this basic function. Also need –, ×, /, & lots more.

LSB = Least Significant Bit

Integrated one-digit binary arithmetic unit (prev. slide)

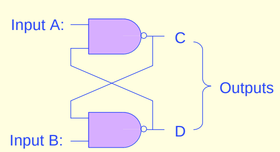
Computer technology built up from pieces

- The foregoing example illustrates the way in which computer technology is built
  - start with little pieces (transistors acting as switches)
  - *combine* pieces into functional blocks (gates)
  - *combine* these blocks into higher-level function (e.g., addition)
  - *combine* these new blocks into cascade (e.g., 8-bit addition)
  - blocks get increasingly complex, more capable
- Nobody on earth understands Pentium chip inside-out
  - Grab previously developed blocks and run
  - Let a computer design the gate arrangements (eyes closed!)

Data Storage

- Within the computer, data is stored in volatile memory (RAM)
  - essentially charge held on a capacitor
  - also possible to rig two NAND gates to hold one bit
    - called a flip-flop
  - volatile because it goes away when turned off
- Also store data permanently, usually on magnetic media (floppies, hard drives, tapes) or on optical discs (CD-ROMs, DVDs)
  - information encoded as polarization of magnetic domains
  - older technology used wire coils around ferrite cores (like transformer) to detect/generate magnetic fields

Example: Flip-Flop Memory



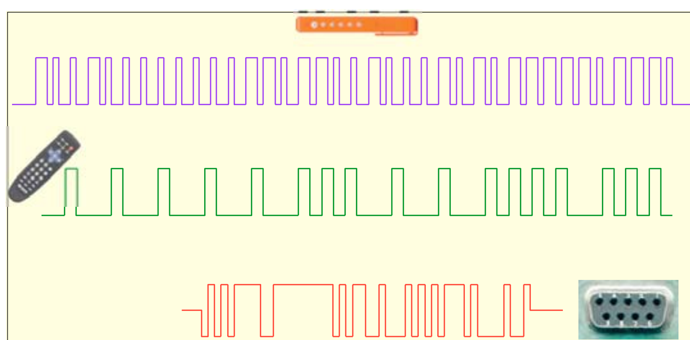
NAND

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

flip-flop

A	B	C	D
0	0	1	1
0	1	1	0
1	0	0	1
1	1	?	?

- This simple arrangement of two NAND gates retains a memory:
- Imagine A and B are in the high state (both 1)
  - C = 0, D = 1 is valid, but so is C = 1, D = 0
  - can set the state by dropping A or B low momentarily
  - when A and B are restored to high, the previous state is "remembered": e.g., B went low → D sticks on 1



Digital Data Everywhere

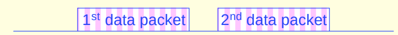
Remote Controls
Computer Communications

Most of today's information is digital

- Most of today's information is digital
  - Computer communications
  - Cell phone signals
  - TV is moving this way
  - TV remote controls
  - Even our beloved in-class infrared transmitters
- Today, we'll look at a number examples
  - start with H-ITT transmitter
  - also check out TV remote (actually for stereo)
  - look at serial data communication

The H-ITT Transmitter Signal

- When you click your transmitter button:
  - A–E: LED indicator comes on, and at same time, TWO bursts of infrared light come out: LED stays on even after transmission stops, until button is released
  - * button: on *release* of button, LED flashes and two infrared bursts are sent

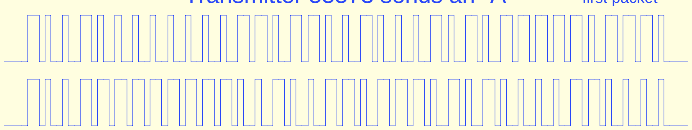


  - bursts last 53 milliseconds, are 9 ms apart, and have a bit-period of about 0.5 ms (about 2000 bits per second)
- Let's look at it on scope…

H-ITT Transmitter Protocol

Transmitter 55573 sends an "A"

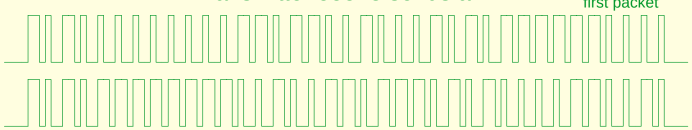
first packet



second packet

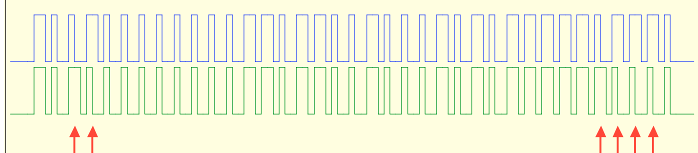
Transmitter 55573 sends a "B"

first packet



second packet

Comparison of A & B first packets



Differences are minor, showing up only near beginning & end

We will represent "high" states (light on) as 1's, and lows (off) as 0's

Notice standard widths: choices are single- or double-width
(both for the zeros and the ones)

Decoding the A signal

Sequence starts out: 0110100100110100100100100 1…

Notice the 01 delimiters: 011010010011010010010010 01…

This gives the signal its choppy appearance (never see 3 1's or 0's in a row)

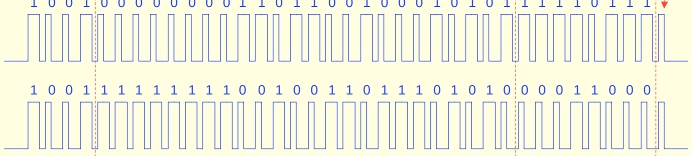
Actual data appears between delimiters: 1's look fat, 0's look skinny

Resulting bit-sequence for A signal (both packets) is:

end delimiter

1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1

1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0

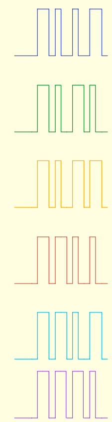


button code

transmitter ID (normal and inverted)

checksum

The different buttons: first four bits



A	1001 → 001 → 1 (first bit always 1)
B	1010 → 010 → 2
C	1011 → 011 → 3
D	1100 → 100 → 4
E	1101 → 101 → 5
<<	1110 → 110 → 6

The Transmitter ID bytes

- Transmitter number is binary-coded in the usual sense:

32768 16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1

0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 0 1

- Sum is:
  - 32768 + 16384 + 4096 + 2048 + 256 + 16 + 4 + 1 = 55573
  - this exactly the number pasted behind the battery
- Second packet inverts all the bits to ensure data integrity

What's with the Checksum?

1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1

button code

transmitter ID (normal first-packet version)

checksum

Break data into chunks of 8 bits (bytes) and add up:

1001
00000000
11011001
00010101
11110111

Checksums provide a "sanity check" on the data integrity

Lecture 12

Another example using newer remote

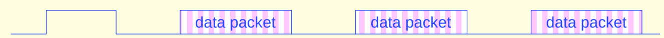
1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 1 0 1 0 0 1 0 0

button code — transmitter ID (normal first-packet version) — checksum

- You can use this (first, non-inverted burst) data to verify your understanding
- Don't look at the answers if you want to challenge yourself first
  - answers on last slide are: button code, transmitter ID, inverted packet contents
  - also check that checksum adds up properly (ignore final "carry" digit)

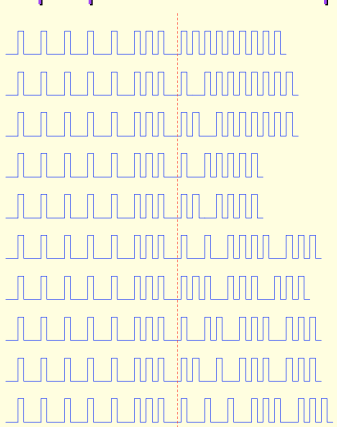
Stereo Remote Control

- Similar to H-ITT transmitters in principle:
  - bursts of infrared light carrying digital information
  - punctuated by delimiters so no long sequences of 1's or 0's
- Key differences:
  - signal initiated by a WAKE UP! constant-on burst
  - pattern that follows is repeated indefinitely until button is released
    - I can never get fewer than three packets…
  - packet is variable in length depending on button



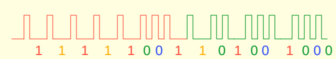
Sample patterns for data packet

Button	Code
POWER	000000000
VOL +	100000000
VOL –	010000000
1	100000
2	010000
3	110001000
4	001001000
5	101001000
6	011001000
7	111001000



A Different Code…

- The radio remote uses a different scheme:
  - does not use the 01 delimiters like H-ITT did
  - instead, uses 10 to represent zero, and 1000 to represent 1
  - sequence for the 5 button is:
    - 1000100010001000100010101000100010100010…



- in data part, least significant bit (LSB) is first
- so the number part of "5" is 101001000 → 1010
- least significant digit is first, so reverse order for more familiar form: 0101 = 5

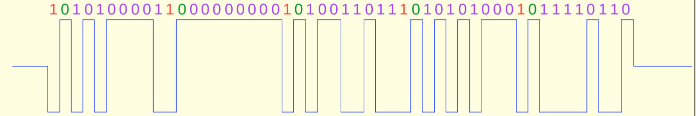
Serial Communication: Getting the Data

- Once the H-ITT receiver gets your IR signal, it must communicate this to the computer
- It does this through the *serial* port
  - serial refers to the fact that data bits arrive in series (one at a time)
  - alternative is parallel (one wire for each bit), where typically 8 bits (a byte) arrive simultaneously
- Most digital communications are of serial type
  - IR transmitters! (only one "channel" for light)
  - USB, Firewire
  - ethernet, modems
  - cell phones
- Parallel sometimes used for printers, but most notably on computer motherboards
  - now 32-bit wide communications is the standard
  - parallel is faster, but more complicated to pull off: lots of wires

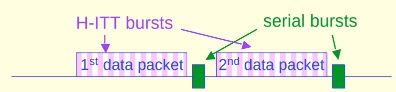
A look at the H-ITT Serial Datastream

E-button on H-ITT (first of two packets):

1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0



- Serial datastream looks a lot different
  - this one allows many zeros or ones in a row
  - delimiters (called start bit and stop bit) bracket 8-bit data (1 byte)
  - in this case, 0's are positive voltage, 1's are negative (backwards!)
  - happens much faster than IR: in this case 19,200 bits/sec (baud)
- Packet breakdown:
  - first packet: button number (5 → E), with LSB first: 101000
  - next three packets are ID, also LSB first within each
  - last packet is checksum type of verification



Wrap-up: Digital Data Everywhere

- Our world now runs on information
  - and most of this is broken down to binary bit codes for transmission, manipulation, storage
- Digital advantage is noise immunity
  - very easy to tell a 1 from a 0, even in the presence of environmental noise

Assignments

- HW 4 due today
- Check website for reading assignments
- HW 5 TBA by end of today
- Q/O #3 due tomorrow (5/12) by 6 PM