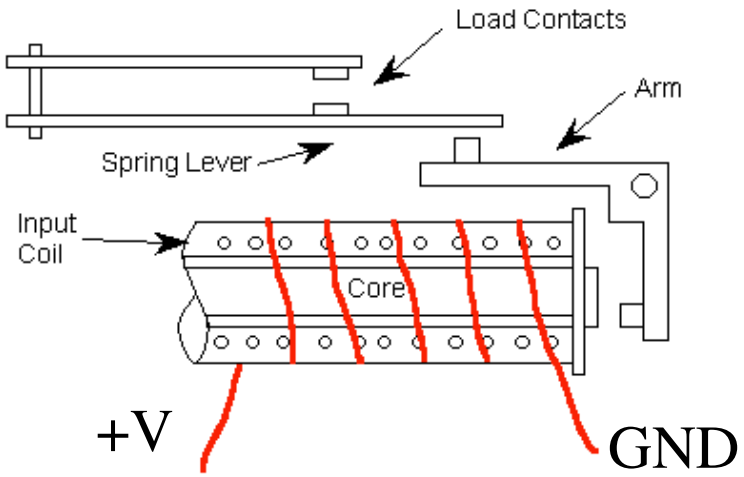
Administrivia

- HW 3 due Thursday
- Milestone 5 due Friday
- HW 4 due next Thursday
- Milestone 6 due Friday

Device of the Day...

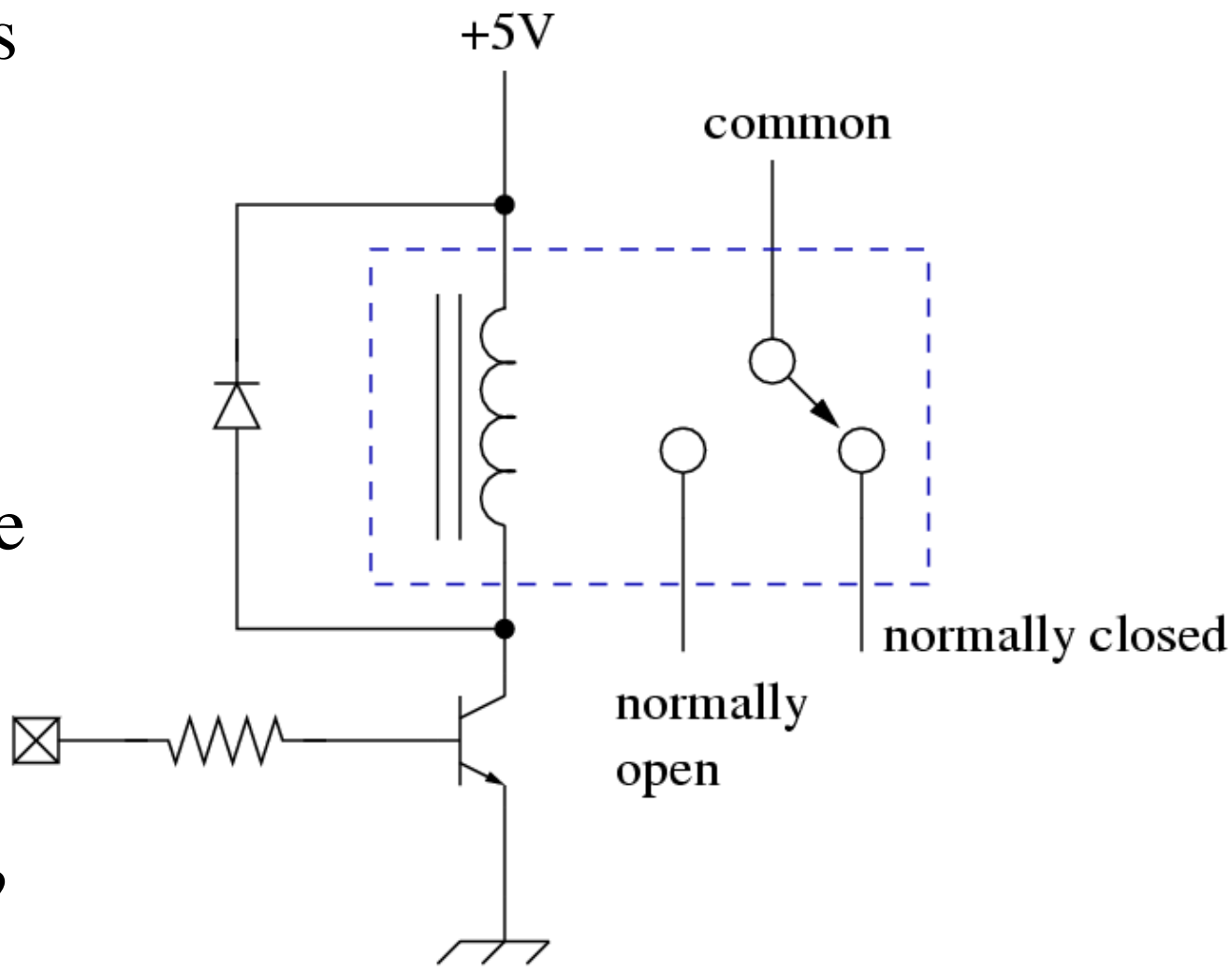
Device of the Day: Electromechanical Relay

- Electromagnet:
  - Iron core
  - Move current through the coil: the core becomes magnetic
- Magnet attracts arm
- Arm closes switch



Driving a Relay with a PIC

- Use transistor as a switch
- SPDT: single pole double throw switch
- DPDT relays are also available (two switches)
- What does the diode do for us?



Driving a Relay with a PIC

What does the diode do for us?

- When a voltage is applied across the coil, it sets up a magnetic field
- Once the transistor is turned off, the field collapses
- This causes a current to flow in the opposite direction (a large, brief current flow)
- Instead of flowing through the transistor (causing damage), the current flows through the diode

Asynchronous Serial Communication

How does it differ from what we have already studied?

Asynchronous Serial Communication

How does it differ from what we have already studied?

- The sender and receiver have their own clocks, which they do not share
- This reduces the number of signal lines
- Bidirectional transmission, but the two sides do not need to be synchronized in time

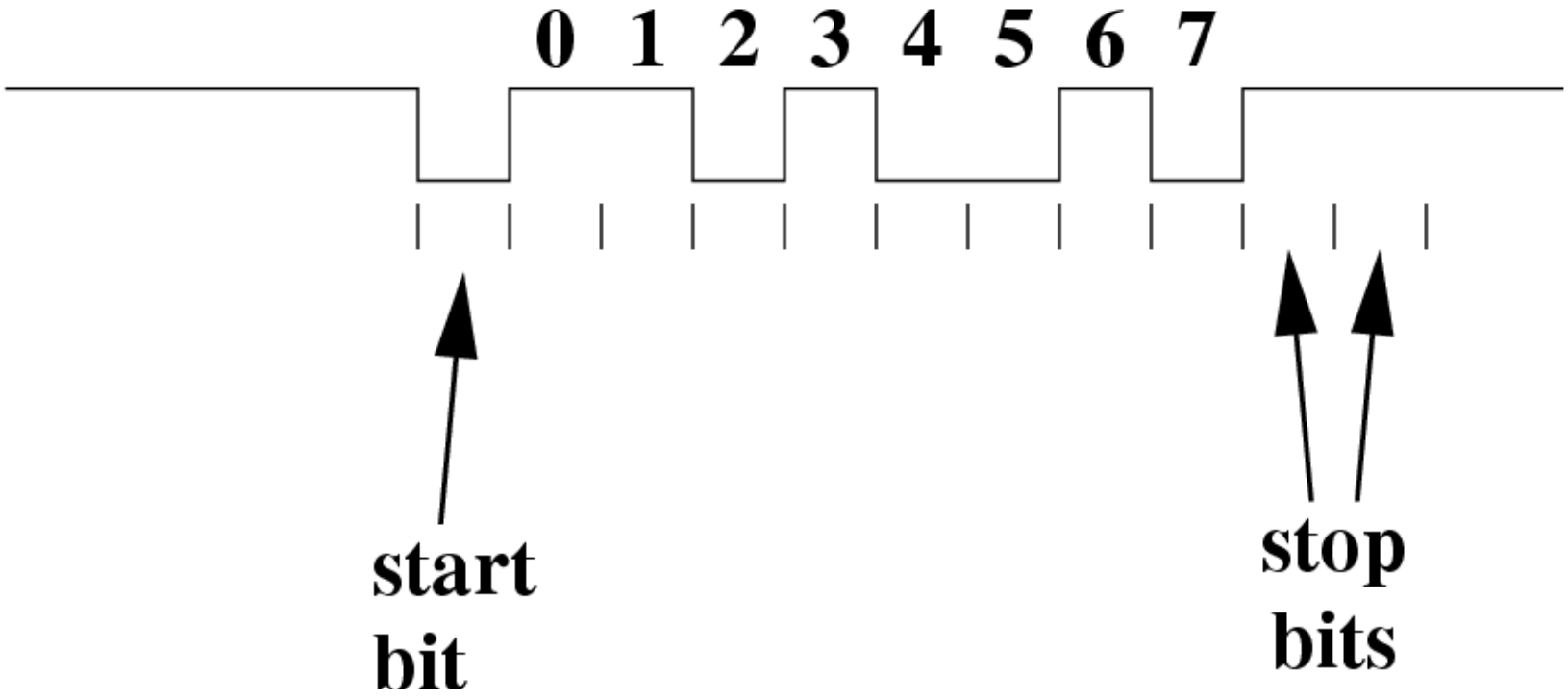
But: we still need some way to agree that data is valid. How?

Asynchronous Serial Communication

How can the two sides agree that the data is valid?

- Must both be operating at essentially the same transmit/receive frequency
- A data byte is prefaced with a bit of information that tells the receiver that data is coming
- The receiver uses the arrival time of this **start bit** to synchronize its clock

A Typical Data Frame



- The stop bits allow the receiver to immediately check whether this is a valid frame
  - If not, the byte is thrown away

Data Frame Handling

Most of the time, we do not personally deal with the data frame level. Instead, we rely on:

- Hardware solutions: Universal Asynchronous Receiver Transmitter (UART)
  - Very common in computing devices (even our PICs)
- Software solutions in libraries (CCS provides software-based functions)

Frame-Level Error Detection

- Due to timing and noise problems, the receiver may not receive the correct data
- We would like to catch these errors as early in the process as possible
- The first line of defense: include extra bits in the data frame that can be used for error detection and/or correction
  - This can be done by our UART

Frame-Level Error Detection

Parity bit: indicates whether there is an odd or even number of 0s in the byte

- Transmitter computes the parity bit and includes it in the data frame
- Receiver also computes parity of the received byte
- If the two do not match, then an error is raised
  - How the error is dealt with is determined by the meta-level protocol

Frame-Level Error Correction

- When we use a single parity bit, we assume that in the worst case, a single bit is corrupted
- But: we can be more sophisticated about catching errors if we transmit more bits (at the cost of a larger data frame)
- Instead, we tend to do these types of checks on a set of bytes (an example in a few slides)

One Standard: RS232-C

Defines a logic encoding standard:

- “ High” is encoded with a voltage of -5 to -15 (-12 to -13V is typical)
- “ Low” is encoded with a voltage of 5 to 15 (12 to 13V is typical)

RS232-C

Originally intended to connect:

- Data Terminal Equipment (DTE)
  - Teletypes
- to Data Communication Equipment (DCE)
  - Modems

Now that we are connecting a computer to some peripheral, it is not always clear which is the DTE and which is the DCE

RS232-C

Defines a pin assignment standard. For example, with the DB-9 connectors:

- Pin 2: receive (to DTE from DCE)
- Pin 3: transmit (from DTE to DCE)
- Pin 5: common

Also common to have DB-25 connectors

RS232-C Handshaking

Flow control is the mechanism by which the receiver can prevent its input buffer from being overrun

- Hardware solution:
  - RTS signal line: transmitter request-to-send: indicates when a new byte is ready
  - CTS signal line: receiver clear-to-send: indicates when the receiver is ready

RS232-C Hardware Flow Control

- Adds 4 more lines to the connector:
  - DB9: pins 4, 6, 7, and 8
- Our PICs:
  - Do not support this functionality in hardware, but it is easy to add
  - These signals will also need to be level-shifted (with the MAX 232)

RS232-C Handshaking

Software solution: dedicate certain data values

- XOFF (^S): stop transmission of data
- XON (^Q): restart transmission

Requires one to take special steps to handle cases where you really want to transmit these values

Leeching Power from the RS232 Interface

- If you are not using the flow control lines (e.g., RTS), one may be used to provide power
- On some computers, you may be able to draw as much as 50 mA (check the specs)
- Make sure the line is set to produce +15V (as opposed to -15V)
- Typically, you will want a voltage regulator (see DEH for an example circuit)

RS232-C is Wonky...

- Different manufacturers make different decisions about which side is DTE and which is DCE
  - Keep those adapters at hand!
- Both sides must agree *a priori* about transfer rates, data frame format, flow control ...
- Cable length limit around 25 meters
- Data rate limits ~38Kbs

An Alternative: RS422

- Also an asynchronous serial mechanism
- A single signal is carried as a voltage difference between two lines
  - Reduced noise due to **common-mode rejection**
- No common ground is necessary
  - Reduction of ground loop problems
- Standard does not define pin assignments
- Cable lengths over a kilometer

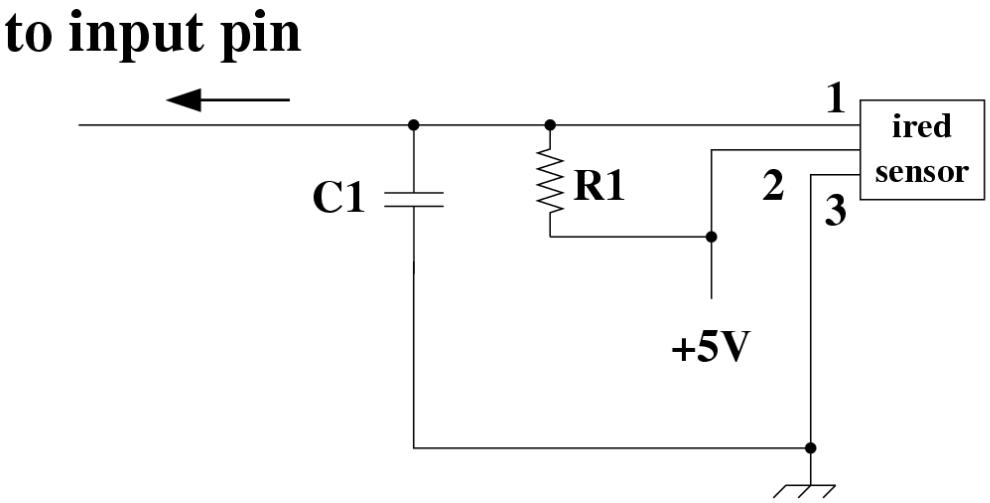
Infrared Communication

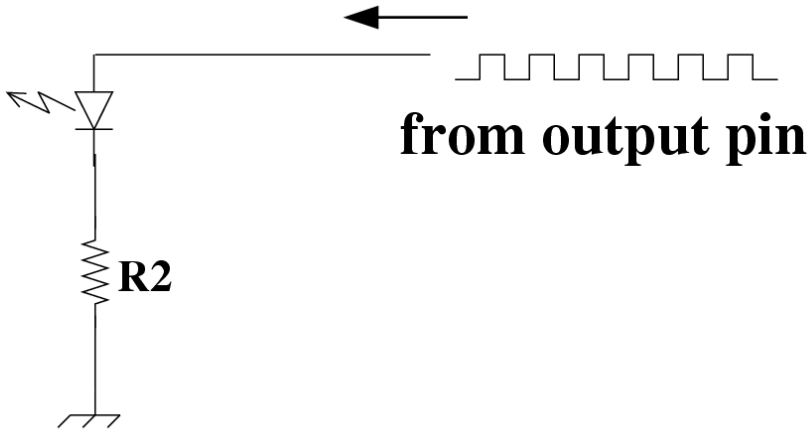
- Directional (10-40 degrees)
- Local (0-4 m)
- Potentially very low power (relative to RF)

How do we generate a signal for the infrared receivers that we have?

One Infrared Serial Implementation

- Recall that our Ired receivers require a 40 Khz carrier
- So – a single bit would be indicated using a sequence of LED pulses





C1: 1μ F
R1: 1 K ohm
R2: 100 ohm (select for your LED)

One Infrared Serial Implementation

- With our Ired sensors, signal reception is indicated as a “low” on the sensor output
- The output of the sensor is Schmidt-triggered
- As the strength of the signal drops (e.g., with distance), the probability of reading a “low” with our PIC drops

How can we be more sure about the true bit value?

One Infrared Serial Implementation

How can we be more sure about the true bit value?

- Sample the state of the sensor many times over the course of a single bit
- Above some percentage of “lows” we will interpret the bit as a “0”

Bit Encoding

Let's make the following (somewhat arbitrary) choice:

- Encode a bit value of “0” with the transmission of an infrared signal
- Encode a value of “1” with no transmission

Infrared Serial Implementation

If our goal is to minimize power usage, how should we structure our data frame?

Infrared Serial Implementation

If our goal is to minimize power usage, how should we structure our data frame?

One possibility:

- 1 start bit of value “0” (why not “1”?)
- 8 bits of data
- 2 “0” stop bits

Could also include a parity bit

Reading a Valid Byte

```
read_next_byte() {
   do {
      wait_for_start_bit();
      for(i=0, out = 0; i < 8; ++i) {
         out = out << 1 + read_bit();
      }
      stop0 = read_bit();
      stop1 = read_bit();
   }while(stop0 == 1 || stop1 == 1);
   return(out);}
```

Reading a Valid Byte

`wait_for_start_bit()`

- Returns at the end of the start bit

`read_bit()`

- Multiple sensor samples
- Returns at the end of the bit

Note that we are throwing out invalid bytes.

How do we handle the timing?

Reading a Valid Byte

How do we handle the timing?

- Engage a timer to tell us when the time for a bit is completed
- Detecting the start bit:
  - Wait for the line to go low; start counter
  - If the line stays low on average, timeout and return
  - Else: go back to waiting

Packet-Level Transactions

Suppose we want to transmit ASCII strings of arbitrary length

- How could we structure a packet?
- How do we make sure that the data arrives intact at the other end?

Packet-Level Transactions

How could we structure a packet?

- A START_PACKET byte
  - Use a non-ASCII value
- The set of data bytes
- An END_PACKET byte
- A checksum byte
  - Use to check the validity of the packet

Receiving a Packet

```
while((value = read_next_byte()) !=
  START_PACKET);
csum = 0; i = 0;
do {
  buf[i] = read_next_byte();
  csum += buf[i];  ++i;
}while(buf[i-1] != END_PACKET
                    && i <= MAX);
if(csum-END_PACKET == read_next_byte()) {
  // valid packet  }
```

Practical Issues

- In order to maximize transmission distance, maximize the current moving through the IRED
- Start with bit rates on the order of 300 bps
- Watch out for interference from other devices
- Don't point the receivers toward the florescent lights

With respect to power, what is the drawback of our approach?

Power Difficulties

With respect to power, what is the drawback of our approach?

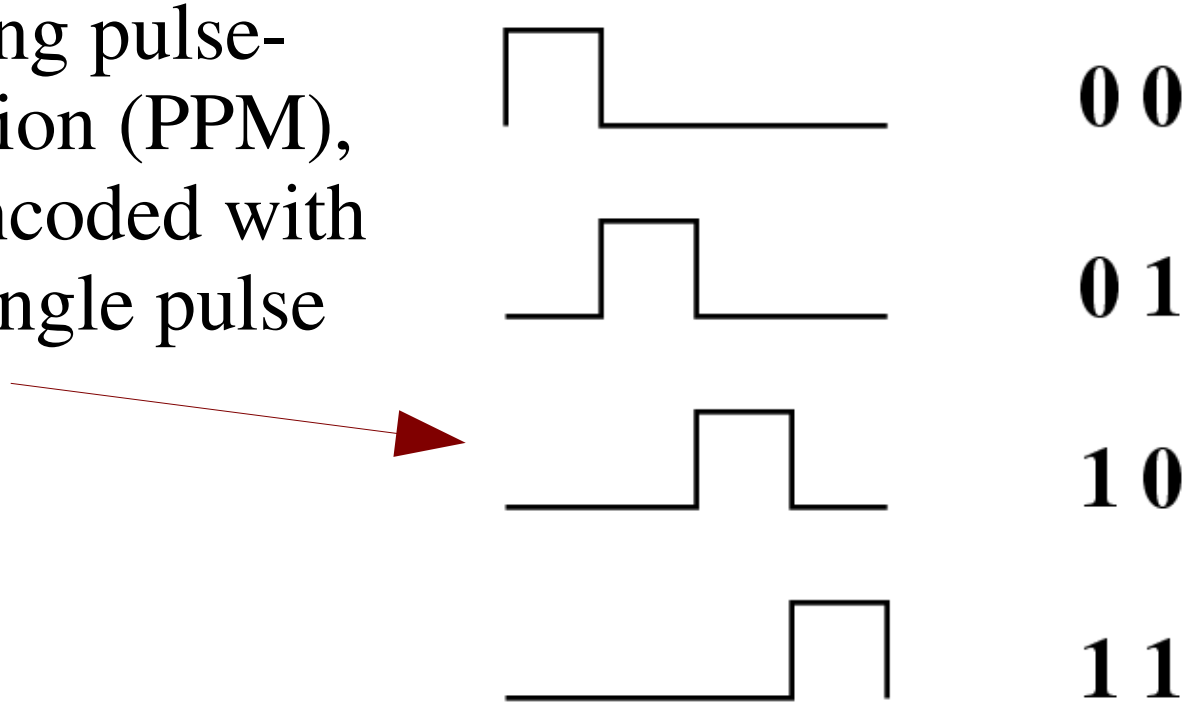
- If we assume a uniform distribution of byte values, our LED is on ¼ of the time
- This means that we are spending a lot of power on transmission

How can we do better?

Power Difficulties

How can we do better?

- Start to be more clever about the encoding of our bits!
- For example, using pulse-position modulation (PPM), sets of bits are encoded with the timing of a single pulse



IrDA Standard

Defines:

- Physical layer (wavelength, transmitter/receiver angles)
- Link access protocol: handles the initiation/termination of a dialog and the flow control issues
- IrCOMM protocol: emulates serial device
- IrLAN protocol: communication with a local area network

IrDA Packet Transmission

- Uses 4-PPM to encode bit pairs
- A packet includes (4 Mbps protocol):
  - Preamble: 16 bytes
  - Start: 2 bytes
  - Data: 2-2050 bytes
  - CRC: 4 bytes (more sophisticated form of checksum)
  - Stop: 2 bytes

Other IrDA Notes

- IrDA-compliant transceivers are available off-the-shelf
- At the beginning of a transaction, the devices negotiate an appropriate data transmission rate
- Inherent assumption that only two devices are involved in a transaction
- Transmission rates up to 4 Mbs

Controller Area Network (CAN)

- Developed in the 80s for use in car control systems
- Now making its way into a wide range of devices (including robots)
- Serial bus standard supporting multiple bus masters
- Wired-AND connection (much like our I2C specification)

Controller Area Network

- Connection standard: uses DB-9 (unfortunately)
  - Ground
  - Optional power
  - CAN_L and CAN_H: differential pair for signal transmission
- Addressing: none.
  - All packets are typed and broadcast. Any module interested in the data may use it
- Bus contention: carrier detect (as with I2C)
  - Packet IDs are selected such that higher-priority packets will naturally win the competition

Other CAN Notes

- Designed for high-noise environments
- 1 Mbs transmission rates
- 1 kilometer max bus length (and modern cars need it!)

Next Time

- Saturday lab open: 2:00
- Lecture:
  - Tuesday: continue discussion about room implementation
  - Thursday: final exam prep and ad-hoc networking