Lecture 22

Wired Communications

Today's Agenda

- Review Wired Communications [Overview]
- UART (RS232,RS485)
- I2C
- Other protocols (I2S, CAN, USB) [Overview]

Concept Design Review – This Friday

- ~8 minute presentation – 7 minute feedback from teaching staff (15minute session)
- Presentation might include:
  - Ideas for the overall approach
  - Critical function – item you might have the most problem with.
  - Long lead time items (e.g. anything you plan to purchase).
  - Schedule for development including consideration for other classes.

Final Project Grading

5%	Design Review 1	Friday April 9
10%	Design Review 2	Friday April 23
50%	Minimum Function	Starts May 4 - Last day May 7

35% Final Report

5% Extra Credit for Peripherals (Lights Sound Action)

Summary Quiz From last time

Select which of the following range sensors is the most appropriate for each case:

a) Retroreflective
b) ZX retroreflective
c) Ultrasonic (ping)
d) PSD Position Sensing Device
e) TOF (Time of Flight)

1. Accurately sense the time for a robot moving in a straight line to hit a wall.
2. Lowest cost way to sense anything 2 cm infront of me?
3. Lowest cost way to sense anything 50cm in front of me?
4. Most reliable way to sense an object approaching 50cm away

01

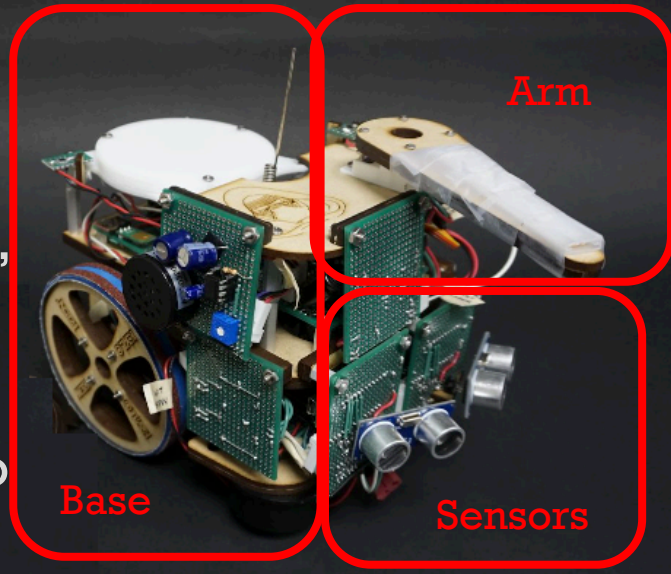
Wired Communication

Quiz questions at end of lecture

1. For asynchronous comm, you must set the same _____ on both ends
2. For synchronous, _____ sets the clock speed (mostly, though I2C has slightly different)
3. _____and ____ are the two most common *synchronous* protocols used between embedded processors
4. _____ can be used for simple limited state changes

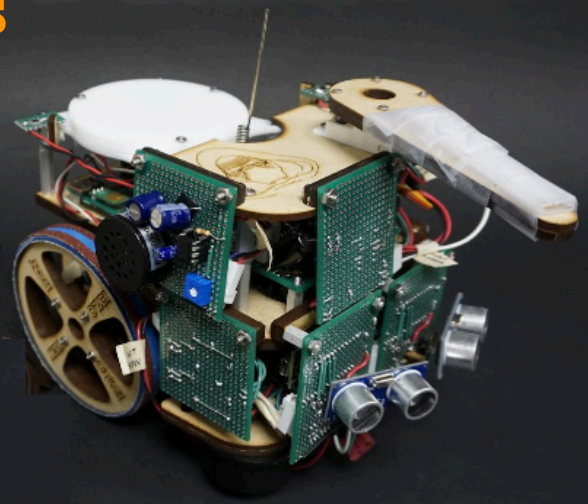
Final Project – Requirements

- Show mode of teleoperation (e.g., tankmode, steering mechanism, holonomic, etc.)
- Move at least one can to doubling circle.
- Move at least two cans from opposing side to score on your side.
- Show at least one autonomous behavior:
  - Wall following, make full circuit
  - Locate and move relative to a beacon or lighthouse



Communications between MCU's

Q1: Draw and hold one line that connects the microcomputers in each subsystem to indicate who needs to talk to whom



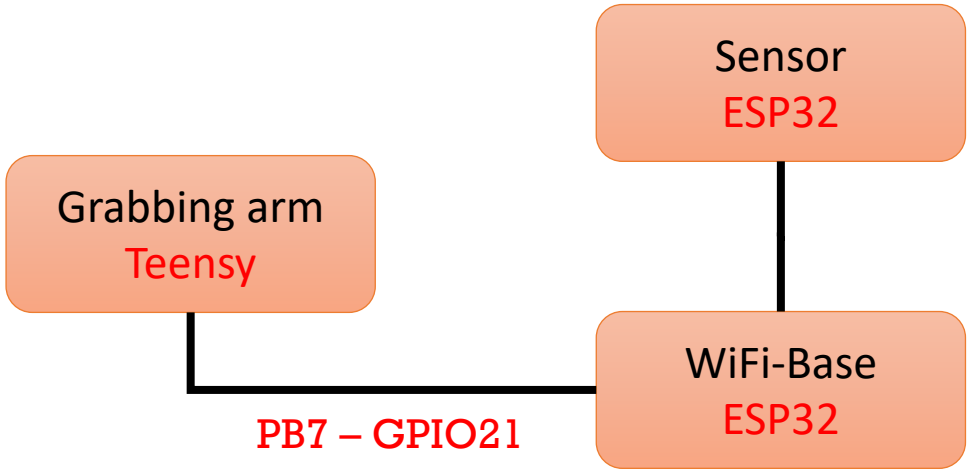
Sensor
ESP32

Grabbing arm
Teensy

WiFi-Base
ESP32

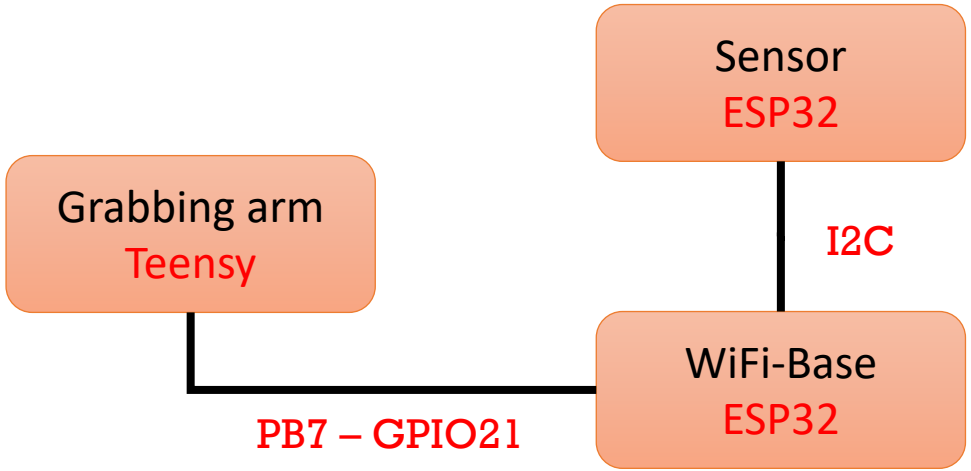
Communications between MCU's

- Connect one output to one input changing logic state e.g.:
  - ESP32 drives LOW to tell Teensy to grab
  - ESP32 drives HIGH to tell Teensy to release



Communications between MCU's

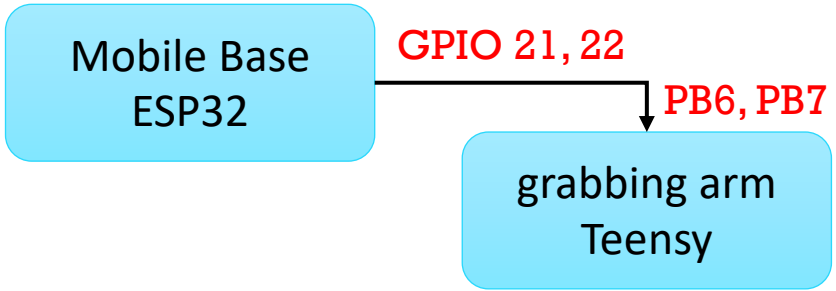
- Example networking of devices
- Usually constrained by subsystems and what pins are available.



Parallel communication

Q3: What signals could we send using two GPIO pins to the grabbing arm in order to do three things:

1. Close gripper,
2. Open gripper,
3. Lift arm holding closed gipper,



Q4: How many states could we control with three GPIO?

Serial Communications

Sequence of bits are used to send arbitrary states (messages) from one to another.

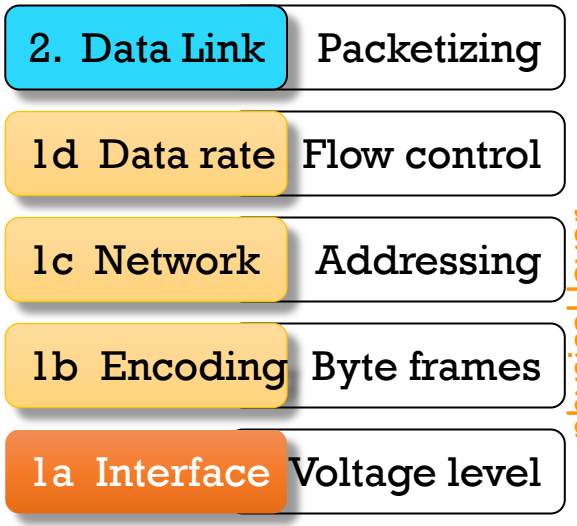
- Transmitter takes bytes (8-bits), transmits one bit at a time.
- Receiver receives bytes of data, reassembles bits into bytes.

Two ways to implement:

1. Hardware module
   - Specialized hardware to automatically assembles bytes without main software control. Runs in parallel like the timer subsystem.
2. Software serial ports
   - Main software drives GPIO (uses external drivers if needed)
   - This is sometimes called *bit-banging.*

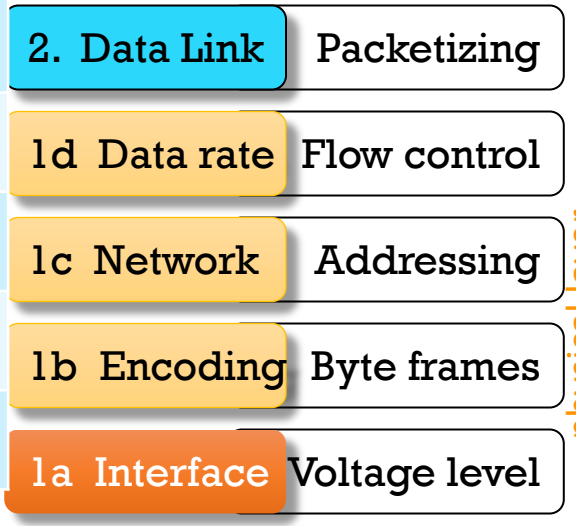
Communications between MCUs and peripherals

- Some protocols specify different components:
  a. **Interface** and voltage levels, sometimes connector physical specs.
  b. **Encoding** bit format and byte frames
  c. **Addressing** multiple devices/multiple masters
  d. **Flow control** (stopping or slowing data rate)
- Can be wired or wireless.



Communications between MCUs and peripherals

CAN	UART Logic	UART RS232	UART RS485	SPI	I2C
x					
		o			x
x				x	x
x	x	x		x	x
x		x	x	x	x



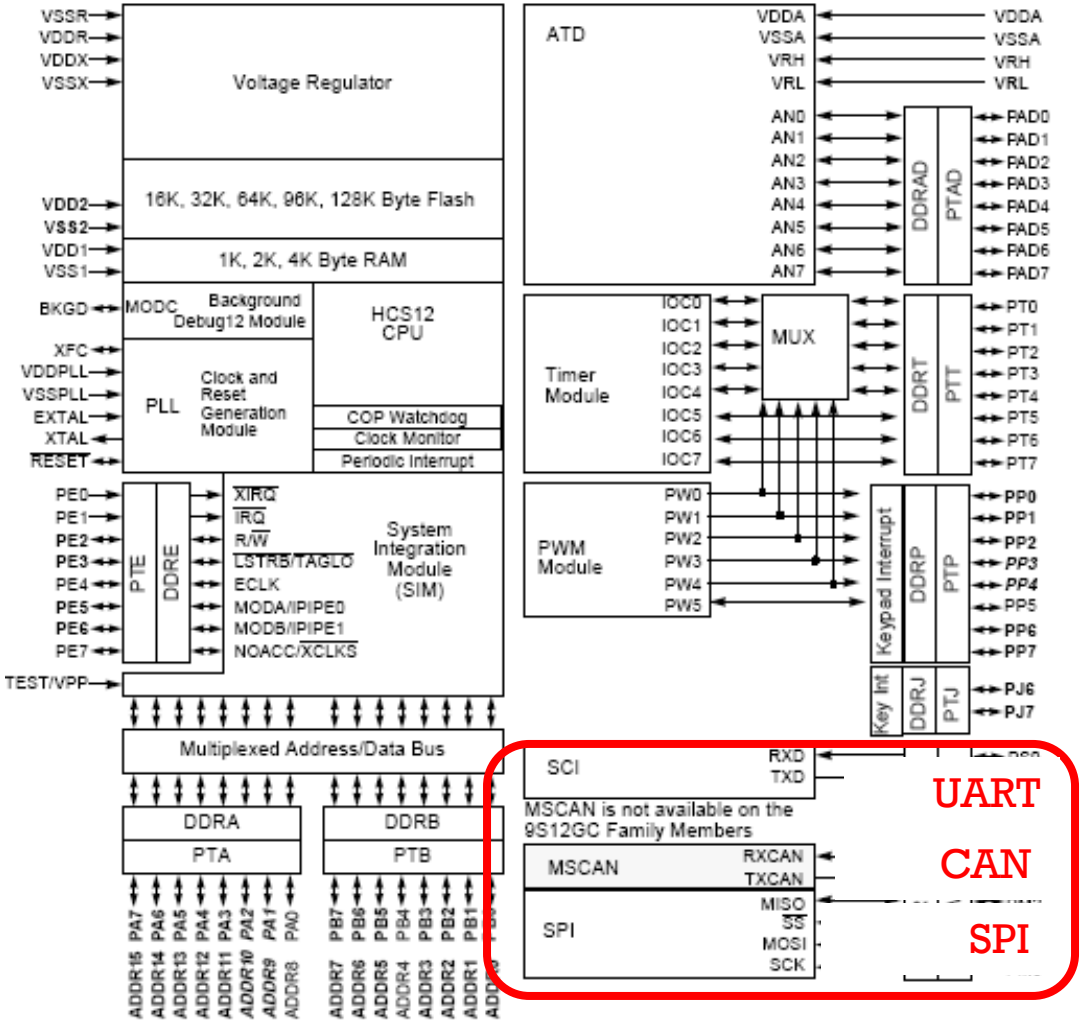
x: protocol specifies

o: protocol has this option (often omitted)

Motorola 9S12

- Similar to 68HC11 family

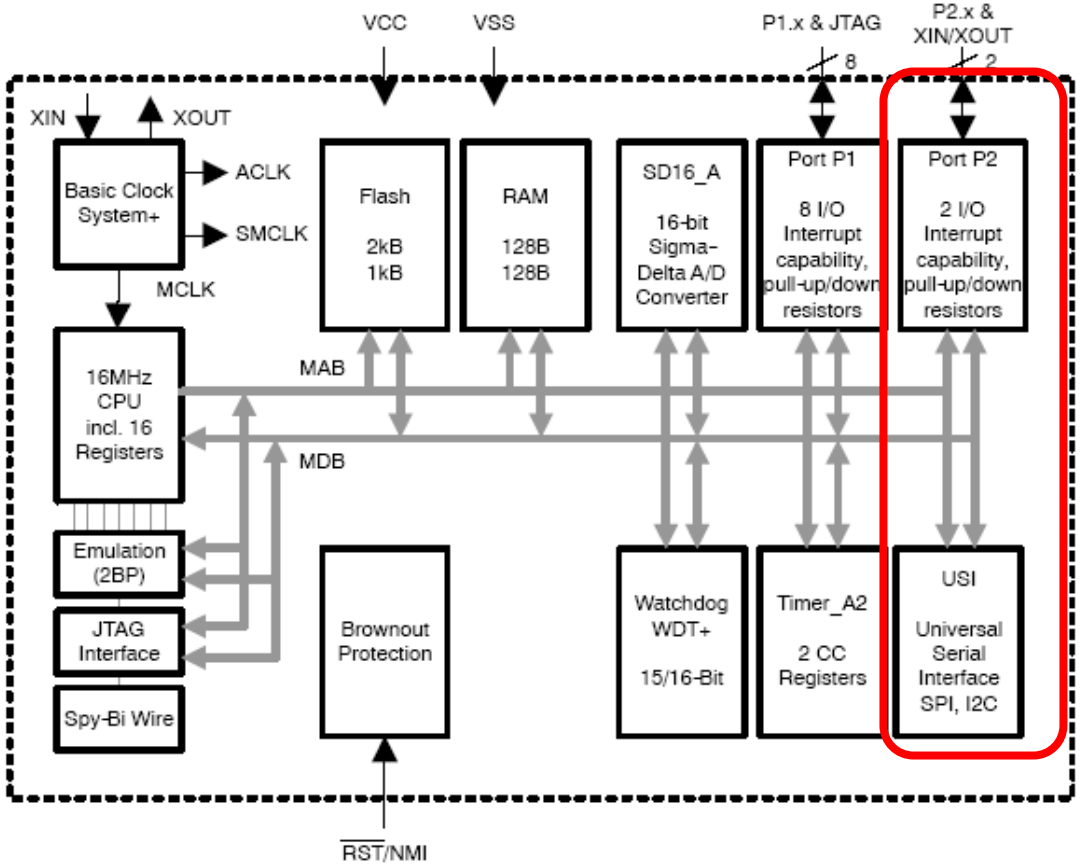
(used in MEAM510 fifteen years ago)



TI MSP430F2013

- Low cost RISC minimalistic

(used in MEAM510 ten years ago)



3 modes on same pins

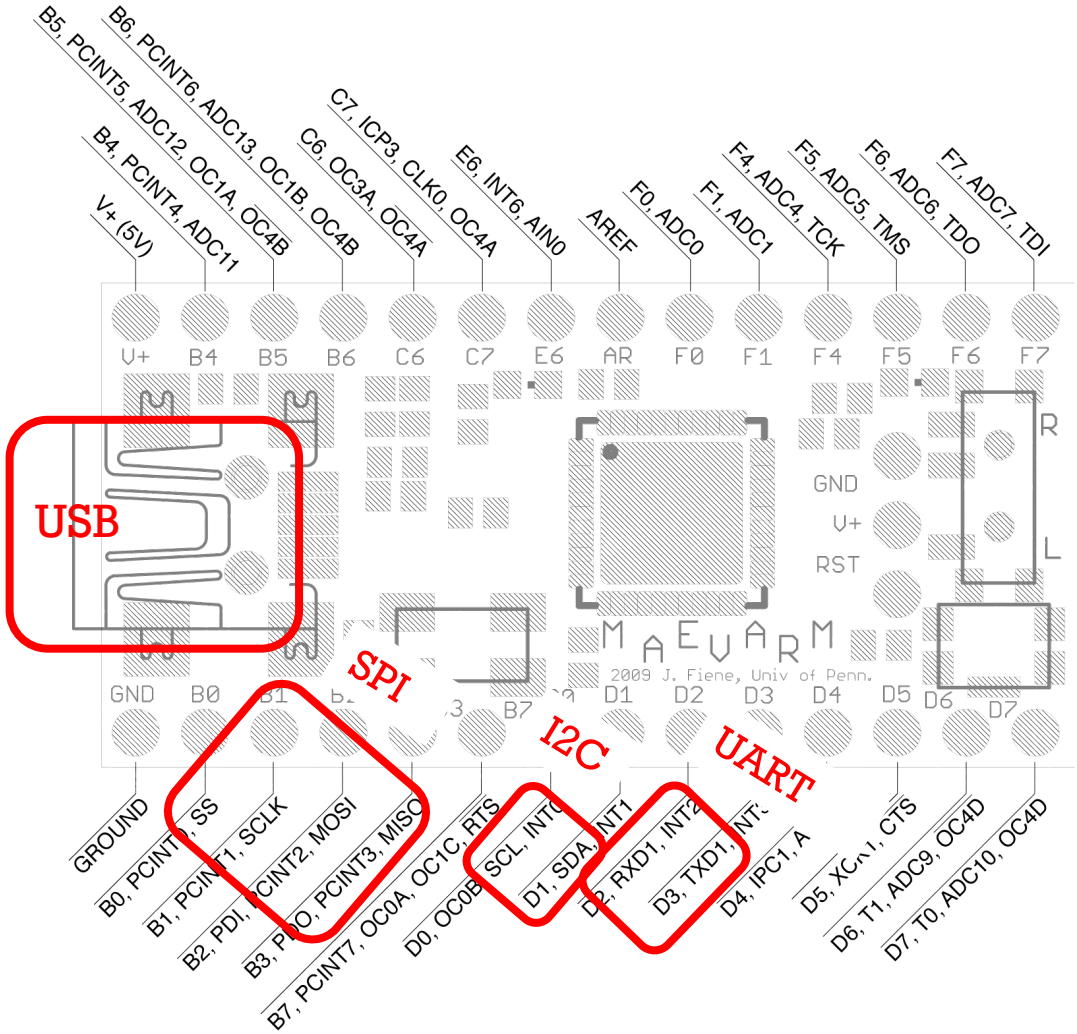
UART

SPI
I2C

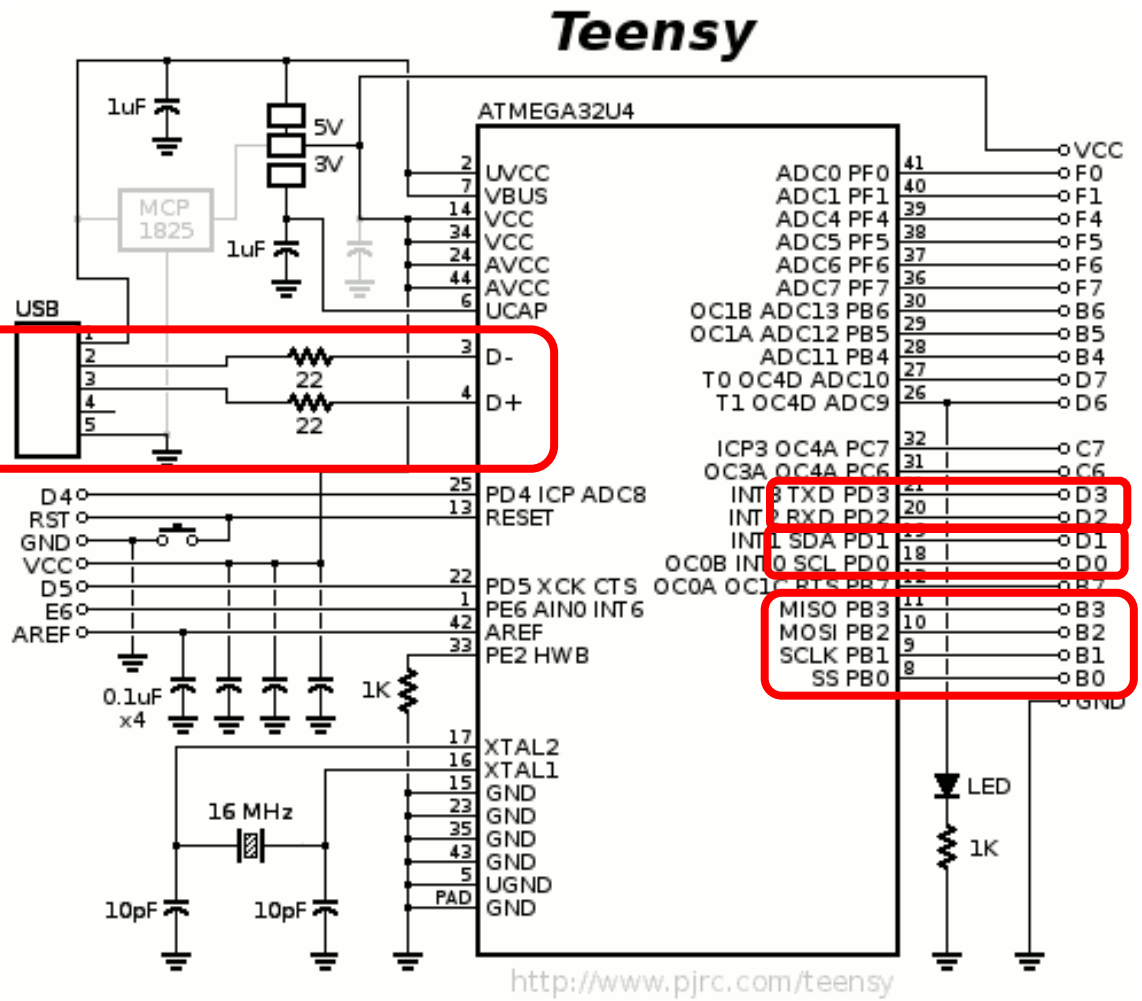
MaEvArM

- ATMega32U4 board
- Very similar to Teensy2.0

(Used in MEAM510 five years ago)



Teensy



USB

UART

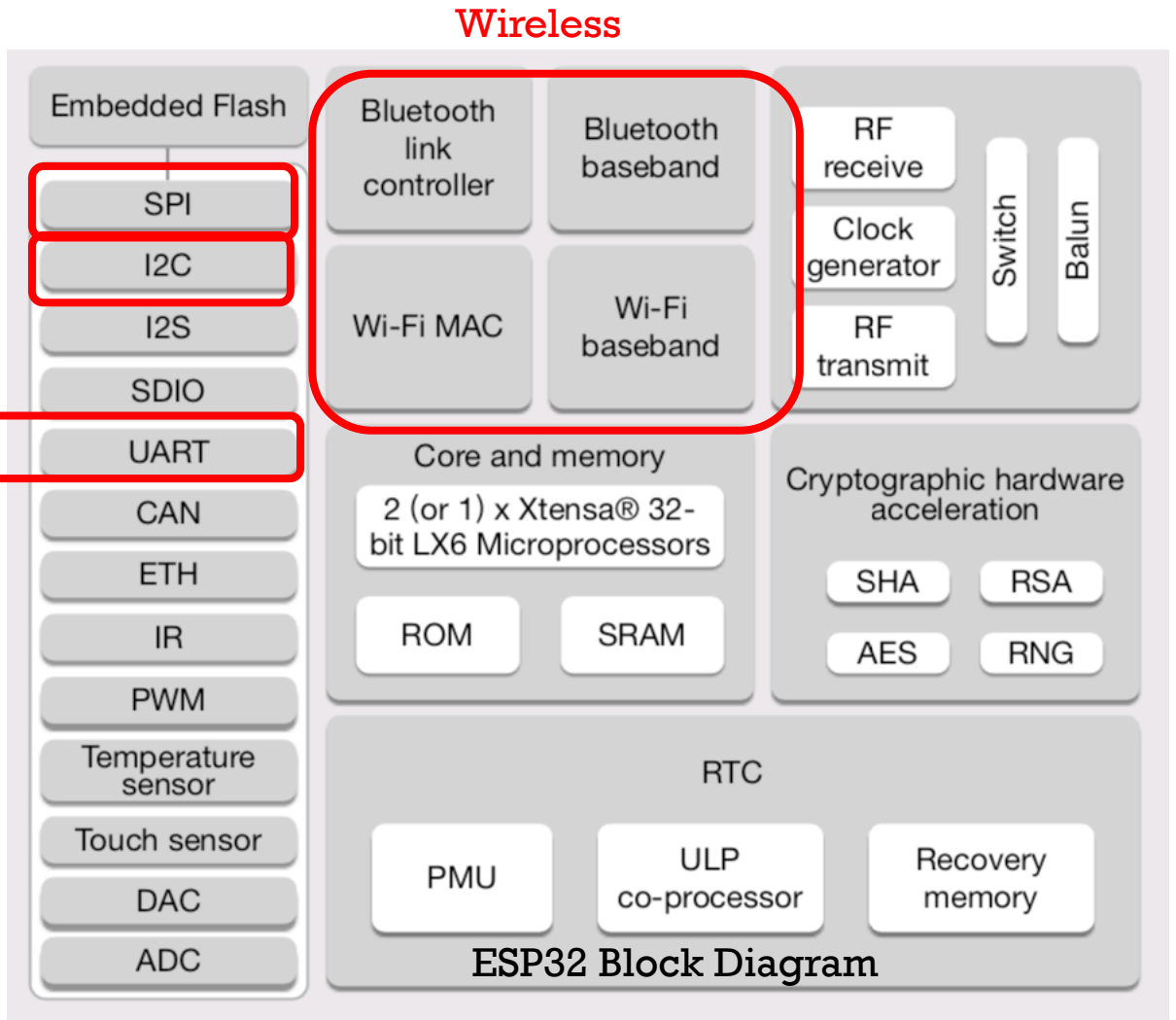
I2C

SPI

PicoKit ESP32

USB to UART

On PicoKit

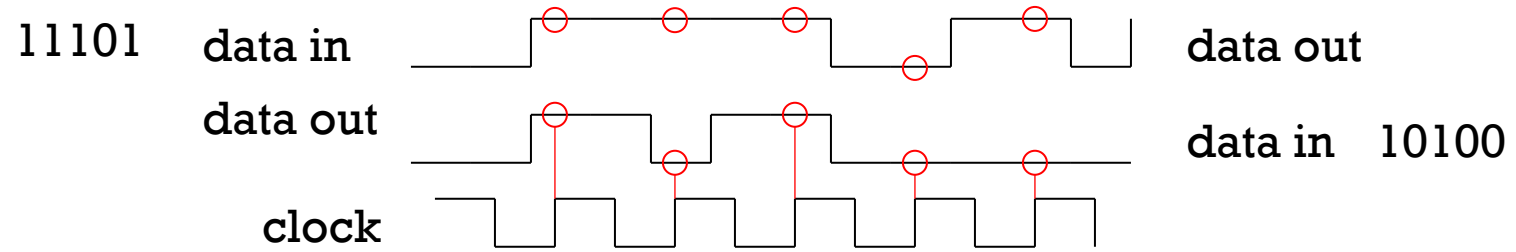


ESP32 Block Diagram

Asynchronous vs Synchronous, Half vs Full Duplex

Several characteristics of serial communications.

- *Synchronous* usually means there is a clock line that indicates when data is ready (e.g. each rising edge of the clock)
- *Asynchronous* usually means no clock line, receiver checks for line state at agreed upon clock rate.



Half duplex for 1 way comm (can switch directions)

Full duplex for 2 way comm

Master and Slave vs Peer to Peer communication

- For master and slave, communications only occur when the master initiates the exchange (whether it's a read or a write)
- For Peer to Peer, anyone can initiate communications usually with a write (send) command.

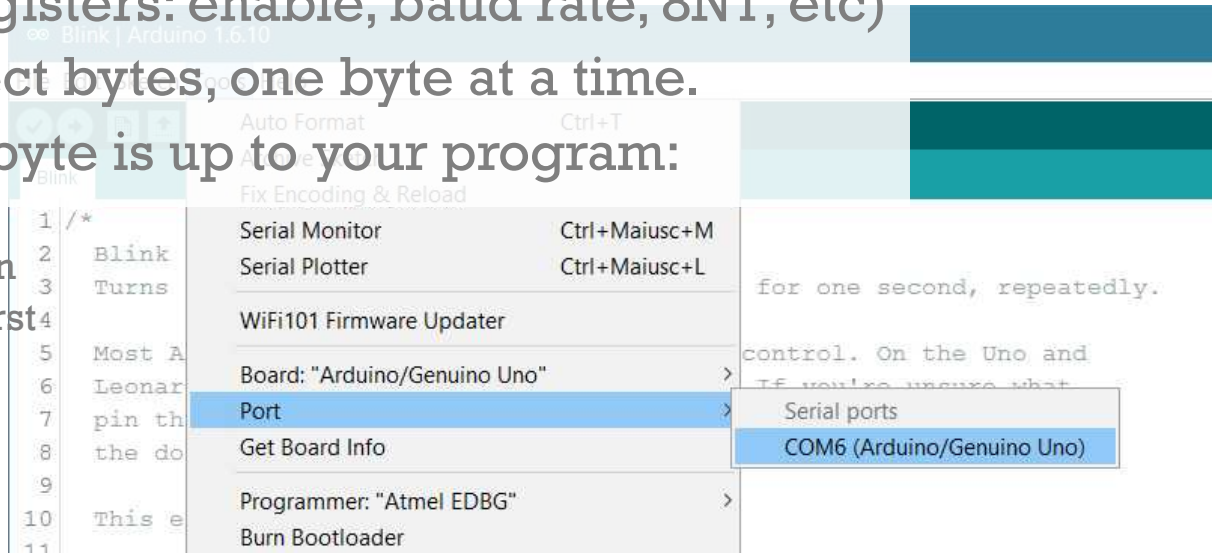
02

UART

Universal Asynchronous Receiver and Transmitter

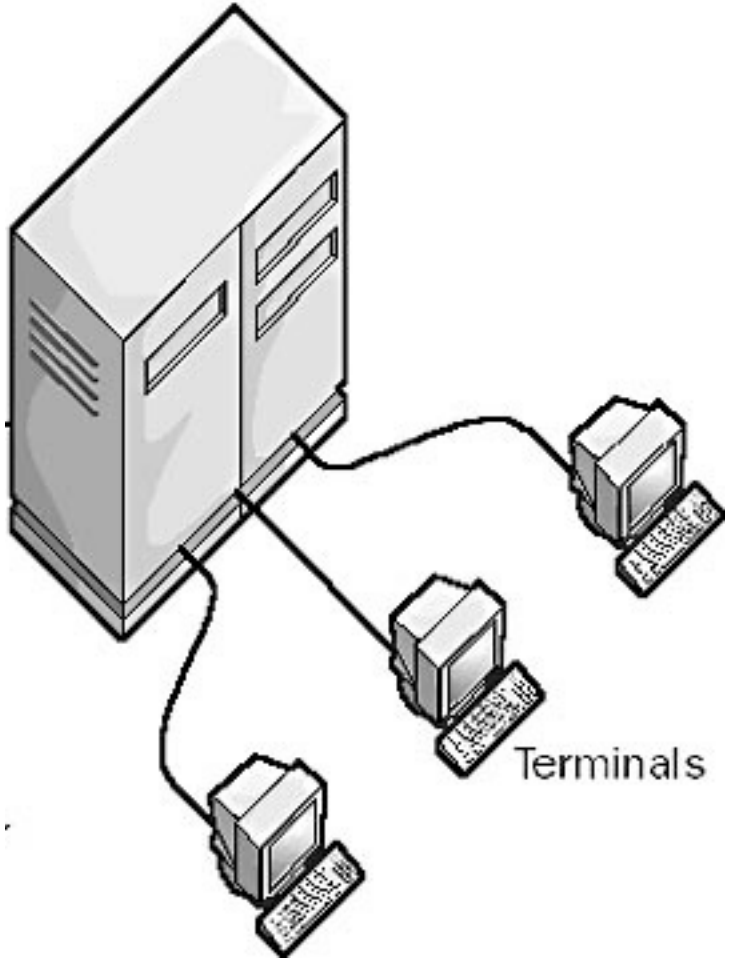
UART: Universal Asynchronous Receiver/Transmitter – aka "serial port"

- Most popular MCU communications method.
  - Slight variant: USART (include synchronous)
- Can use logic levels or include interface layer, typ: RS232, RS422, R485.
- Initialization (setup registers: enable, baud rate, 8N1, etc)
- UART's will send/collect bytes, one byte at a time.
- What you do with the byte is up to your program:
  - Byte order
    - Big-endian, little-endian
    - LS byte first, MS byte first
  - Use Packets?
  - Do handshaking?



Serial port history (all async)

- 1970's Mainframe and terminals.
  - All connected with RS232 serial ports running 9600 baud
- 1980's IBM PC's and knock offs
  - All personal computers had RS232 ports (called COM ports)
  - All running at 9600 baud default, 115200 baud max.
- 1990's USB
  - Serial ports started to disappear.
  - Most USB still had UART converter still called COM ports on PC's

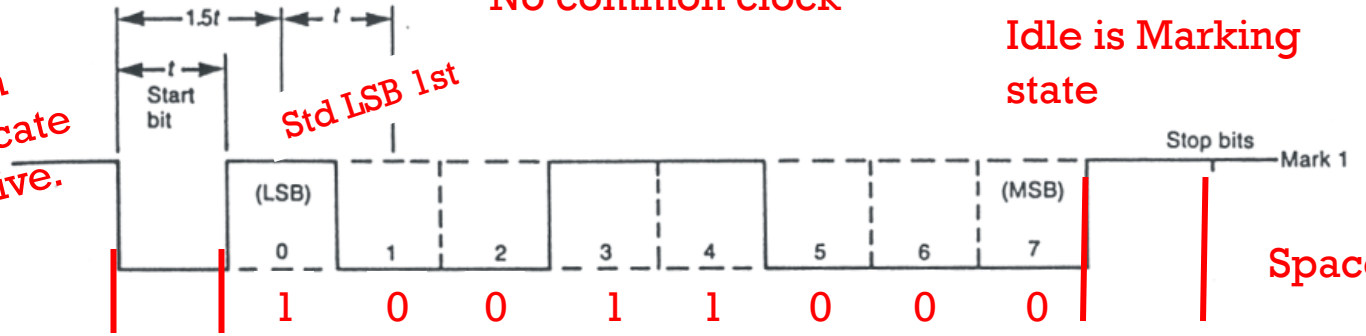


Asynchronous Framing
NRZ (non return to zero)

time

No common clock

Historically nominal high state to indicate device is alive.



Idle is Marking state

Space state

1 bit time

Signifies beginning of byte

Usually 8 bits, Sometimes 7, 9

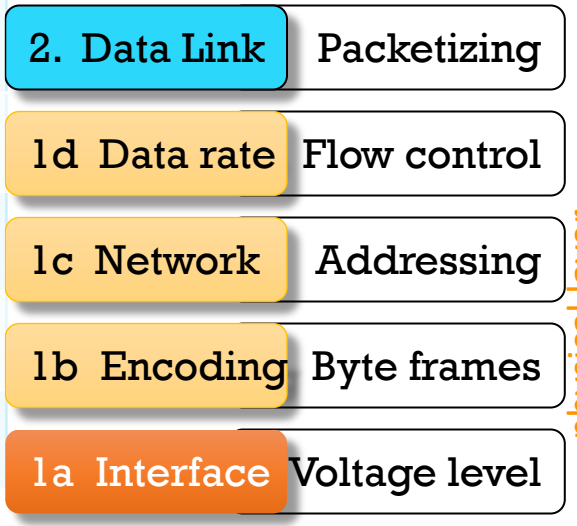
At least one Stop bit

8N1 : 8 Data, No parity, 1 stop bit
8N2 : 8 Data, No parity, 2 stop bit
7E1 : 7 Data, Even parity, 1 stop bit
7E2 : 7 Data, Even parity, 2 stop bits

Baudrates are the bits/second, not data which has overhead (10bits for every byte), Most common: 9600 and 115.2Kbaud.

Communications between MCUs and peripherals

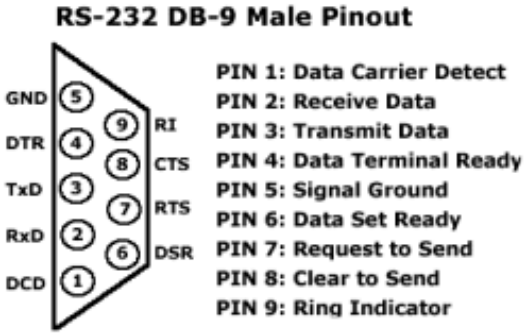
CAN	UART Logic	UART RS232	UART RS485	SPI	I2C
x					
		o			x
x				x	x
x	x	x		x	x
x		x	x	x	x



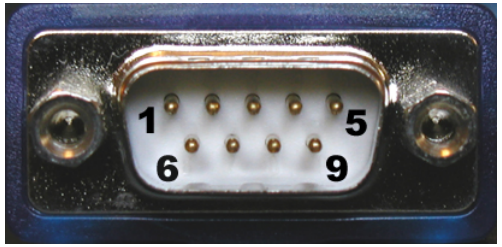
x: protocol specifies
o: protocol has this option (often omitted)

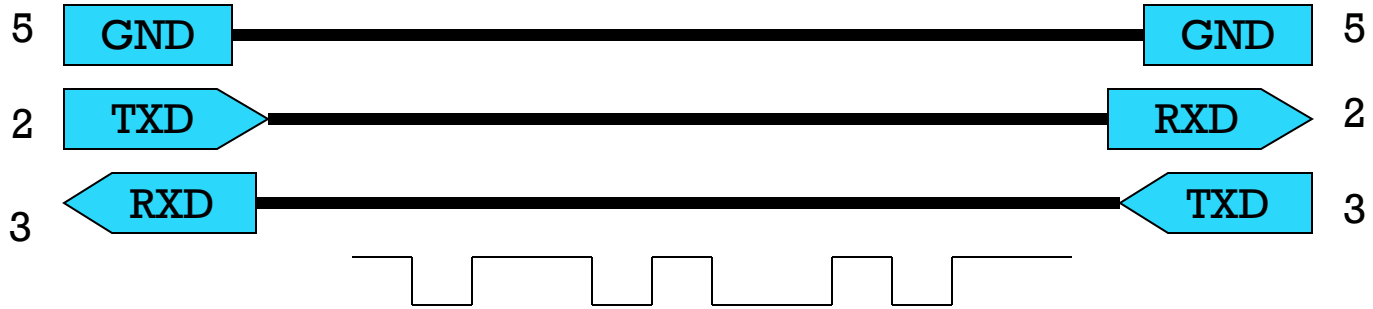
Asynch. Serial Communication– RS232

- RS232 used to be most popular form of inter-processor standard, now RS232 form factor is almost obsolete.
- Most common is essentially 3 wire communication



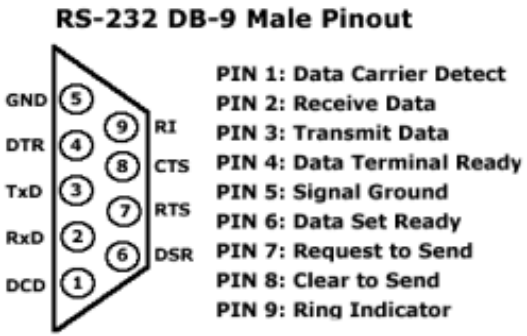
Female is reversed
Numbers start 1 on right side



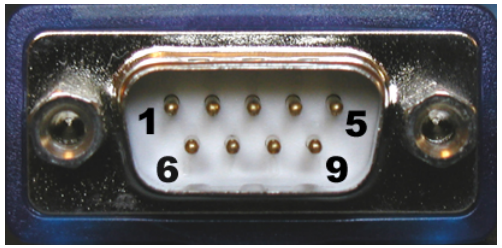


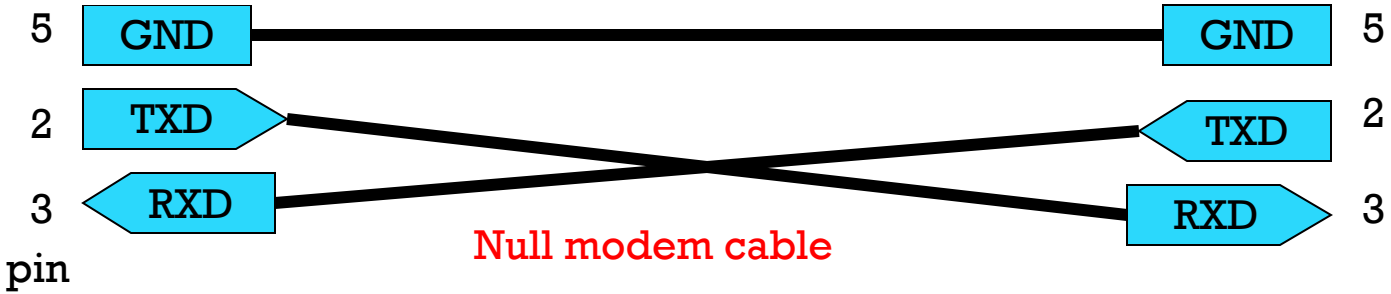
Asynch. Serial Communication

- RS232 used to be most popular form of inter-processor standard, now RS232 form factor is almost obsolete.
- Most common is essentially 3 wire communication

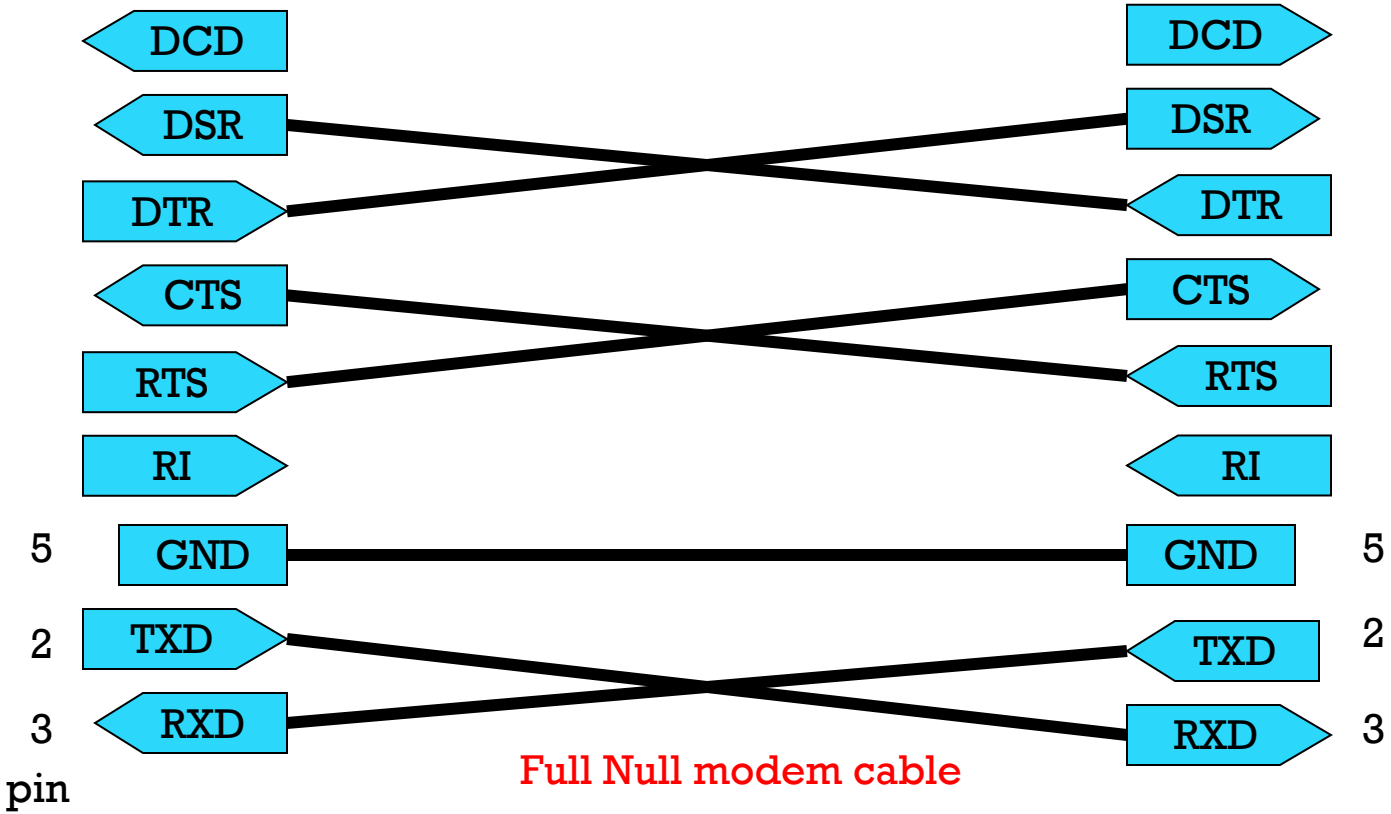


Female is reversed
Numbers start 1 on right side

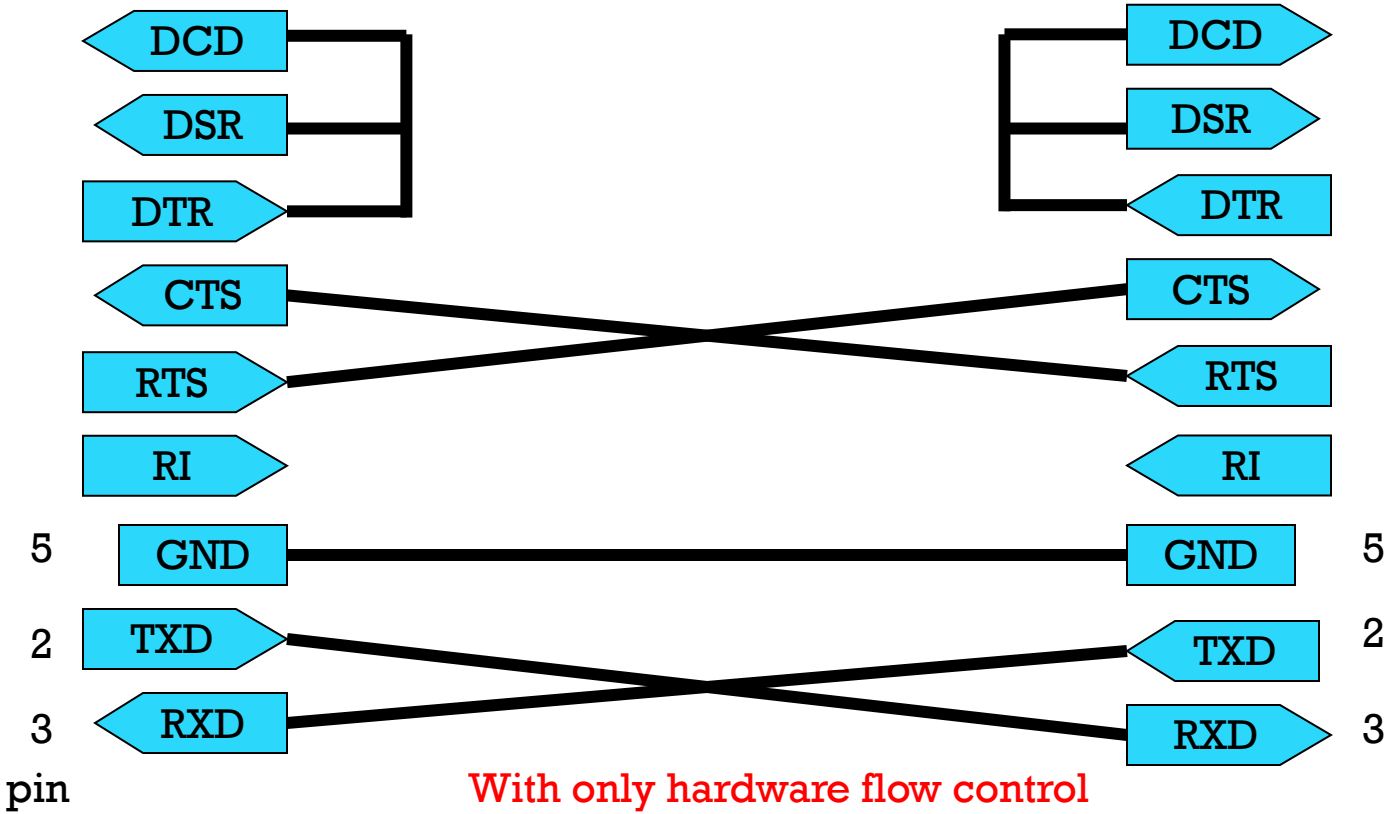




Asynch. Serial Communication (RS232)

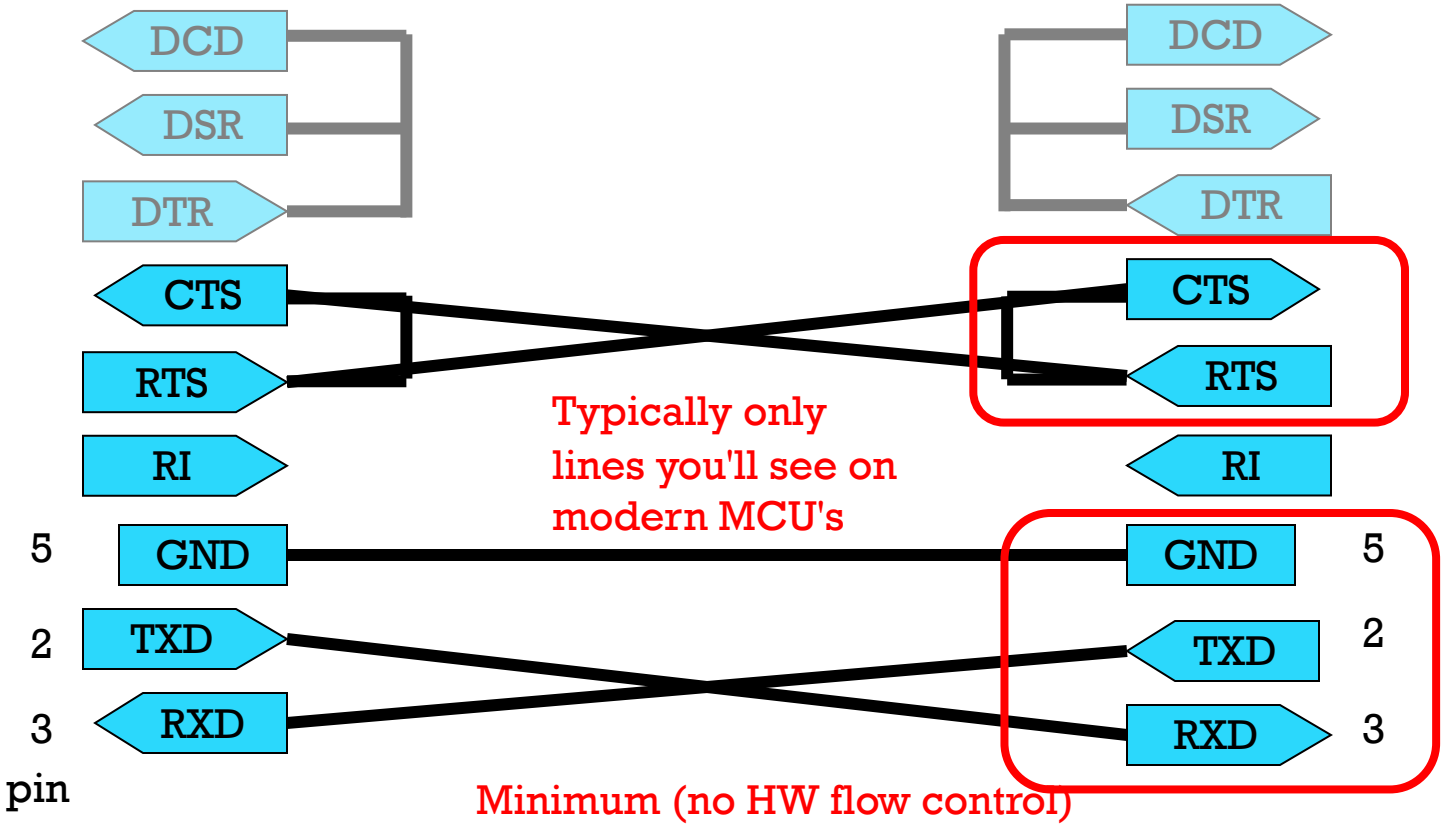


Asynch. Serial Communication (RS232)

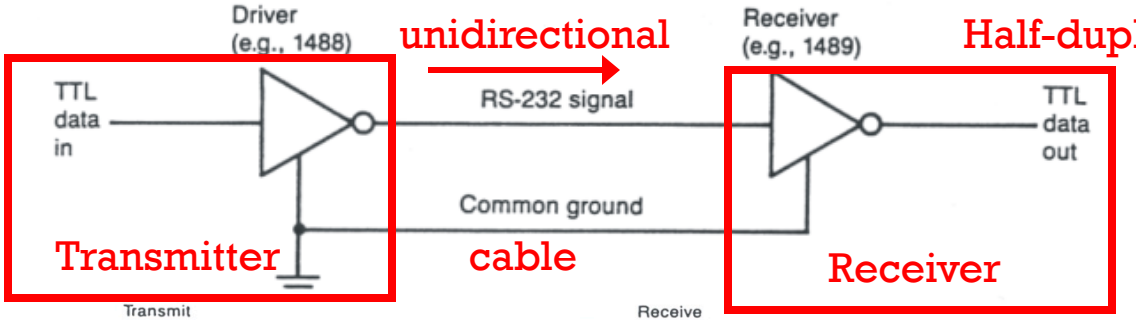


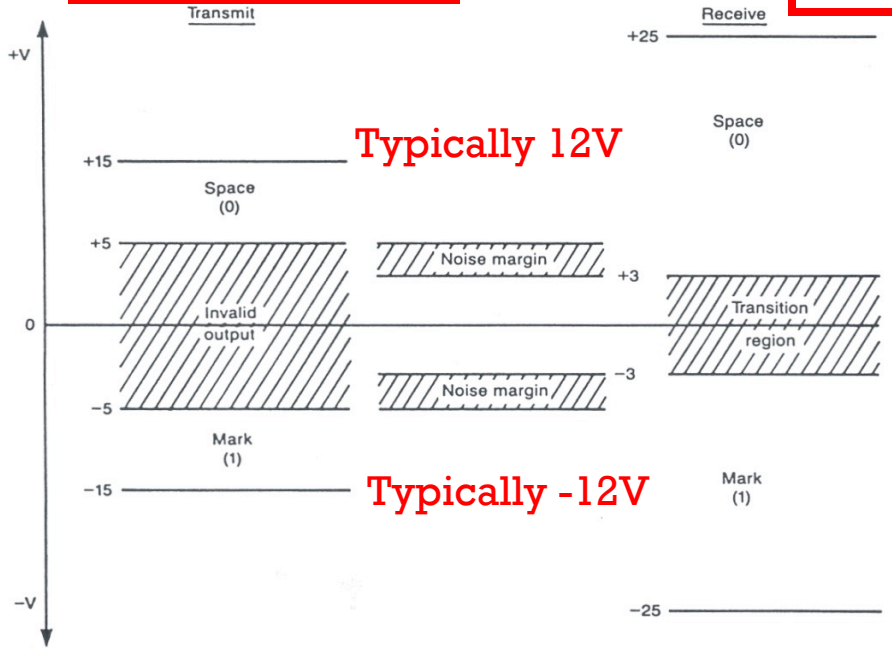
With only hardware flow control

Asynch. Serial Communication (RS232)



Physical Layer – RS232



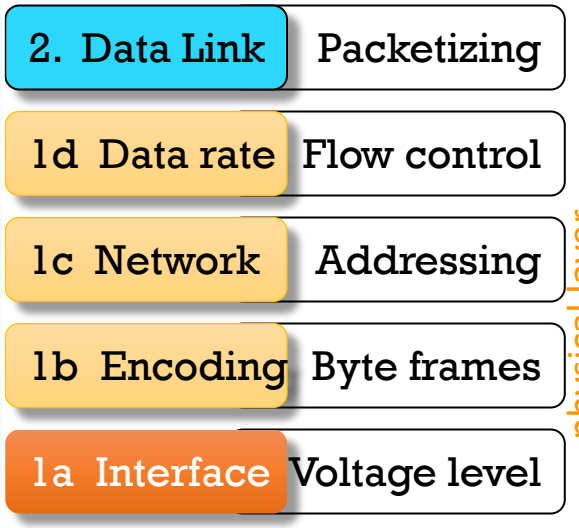


UART out of MCU is usually logic level (not +-15V required for RS232)

Need RS232 drivers.

Communications between MCUs and peripherals

CAN	UART Logic	UART RS232	UART RS485	SPI	I2C
x					
		o			x
x				x	x
x	x	x		x	x
x		x	x	x	x

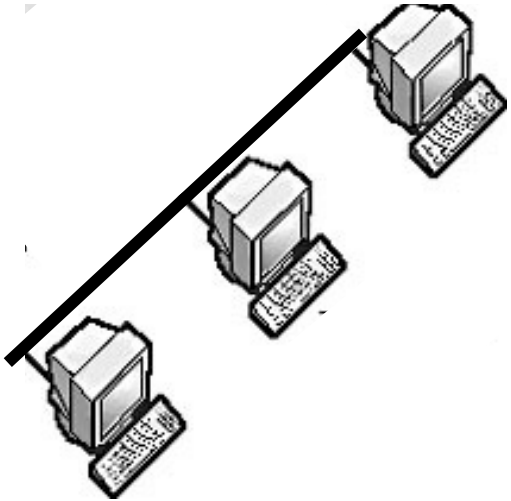


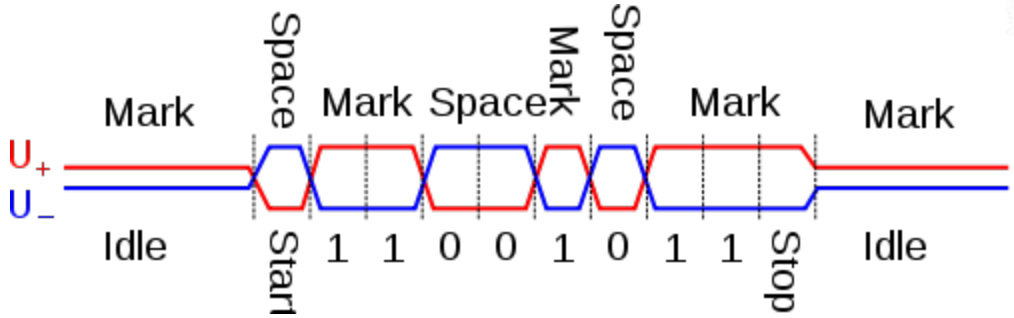
x: protocol specifies

o: protocol has this option (often omitted)

Physical layer – RS485 – ex: differential drive

- Multi-drop
  - (no protocol at this layer – need for collision)
- Uses twisted pair for each signal
  - Robust to noise
  - Implicit gnd, 3 line half-duplex, 5 line full duplex
- Caveats
  - Terminating lines can help robustness





03

Synchronous Comms

I²C

SPI [next lecture]

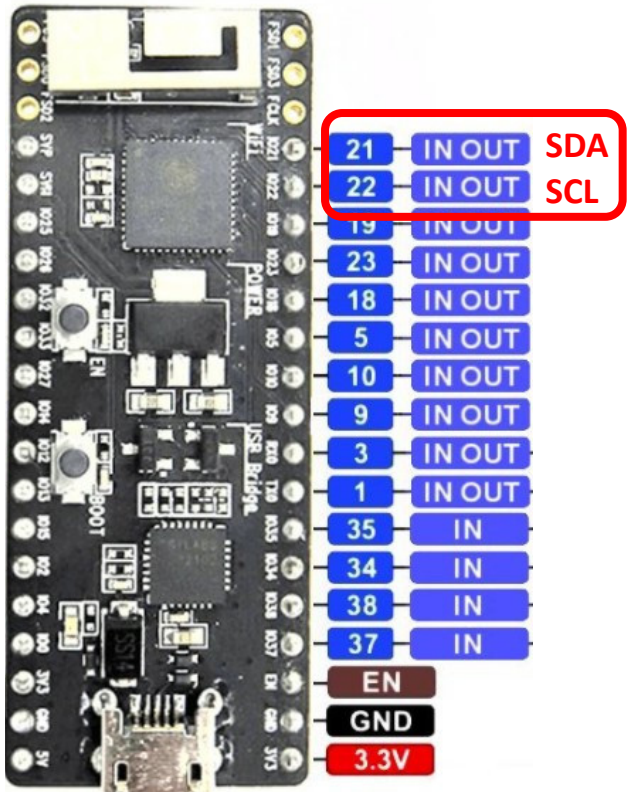
Synchronous Serial Communication

- Synchronous means that there is a line dedicated to indicate when data transfers between all senders and receivers occur.
- Usually this line is called the CLOCK line or CLK.
- SPI and I2C are the two most common synchronous communications protocols between two processors
- Both are very similar except:
  - I2C is half duplex. SPI is full duplex
  - I2C can use as few as 2 lines, SPI needs 4 lines, (possibly 3)
  - I2C max speed is slower than SPI max speed
  - I2C can be made lower cost than SPI
- We will cover SPI in the next lecture.

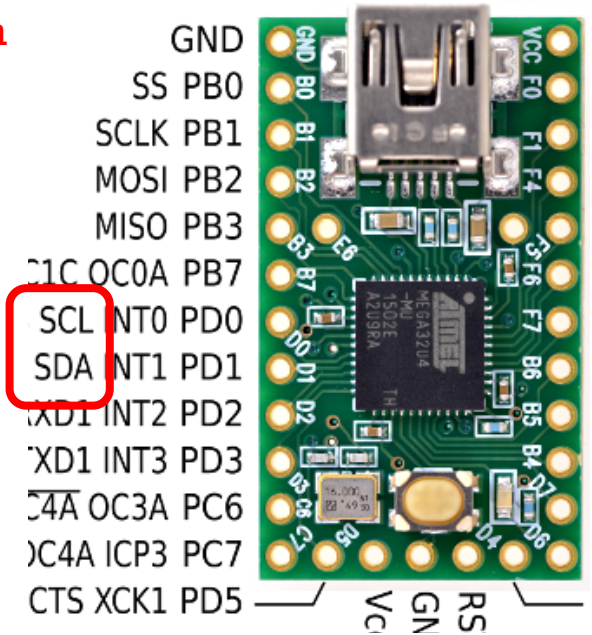
I^2C (Inter-Integrated Circuit)

- Invented by Phillips (NXP)
  - Pronounced "I squared C" sometimes I-two-C
  - Inter-IC implies between chips on a board, not cabled.
- 2 wire (plus ground). Sometimes called Two Wire Interface.
- Synchronous, Half-duplex
- Multidrop – multimaster – 7 Bit address space. (~100 devices)
- Higher level protocol than SPI (packets-switched)
- Typically 40,000 bytes per sec speed (400Kbit/sec)
  - Newer high speed modes – 1Mbit/sec – 3.4Mbit/sec with special hardware drivers

I2C on Arduino – "Wire" Library (not quite done for ESP32)



According to doc: can be set to any GPIO pin (probably not 34+)

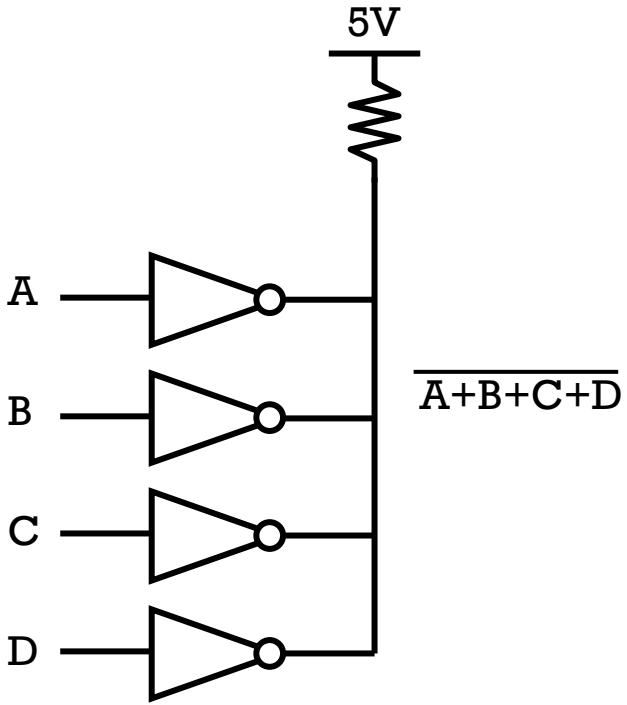


https://www.pjrc.com/teensy/td_libs_Wire.html

Multidrop Wired Bus

- Simplest form of multidrop network
  - "Wired-OR" (low if any is output is low)
  - Broadcast one output to many inputs
  - All drivers use open collector or open drain
  - What limits # of drops on this network?
- Simplest form of multimaster network
  - No added drivers
  - Very low cost (just add pull up resistor)
  - Collisions are safe
  - I2C uses wired-or

Q5: What is potentially wrong with this circuit?

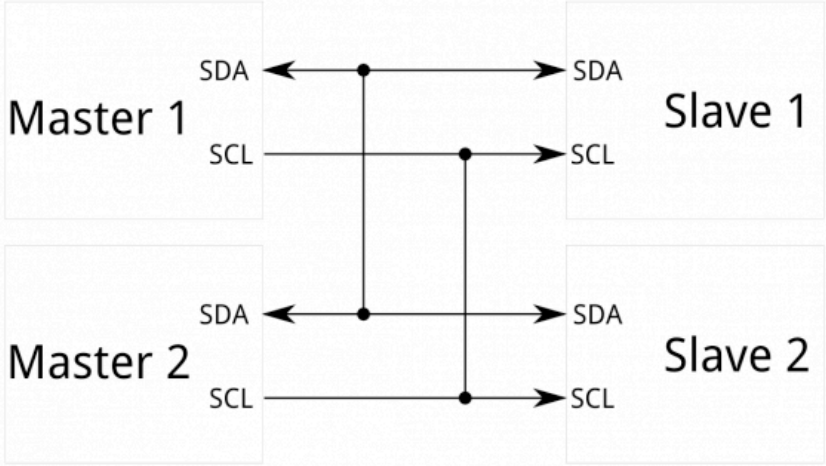


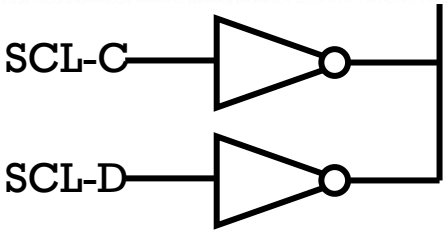
Open Collector/Open Drain output

I2C Open-Drain (wired-OR)

- Two wires (plus ground)
  - SCL (clock line)
  - SDA (data line)

- Master drives Clock line low, but slaves can 'stretch' clock by also pulling SCL line low to slow it down.
- Can easily use 3.3V or 5V as pull–ups can float to any V





Ex: 4 devices driving SCL line

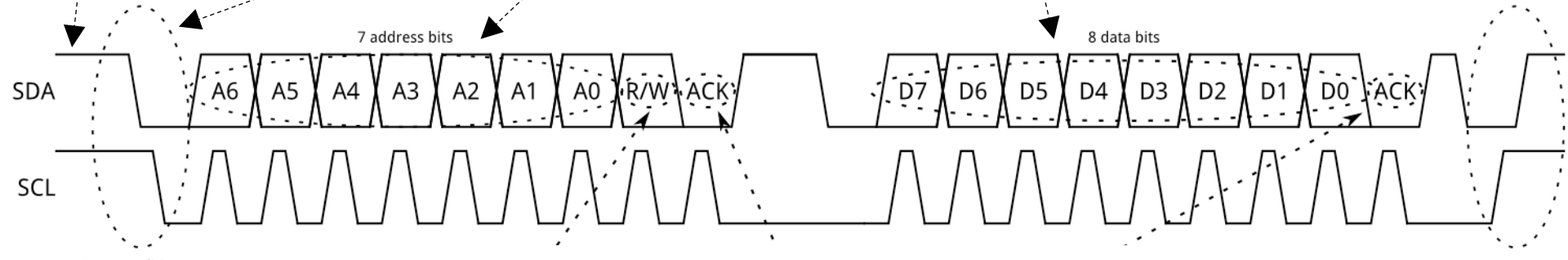
I2C Line Protocol

Nominal state, all lines float high

Master Asserts Start condition

Master drives sequence for address of target slave

Data is transmitted Direction (depending on read/write)

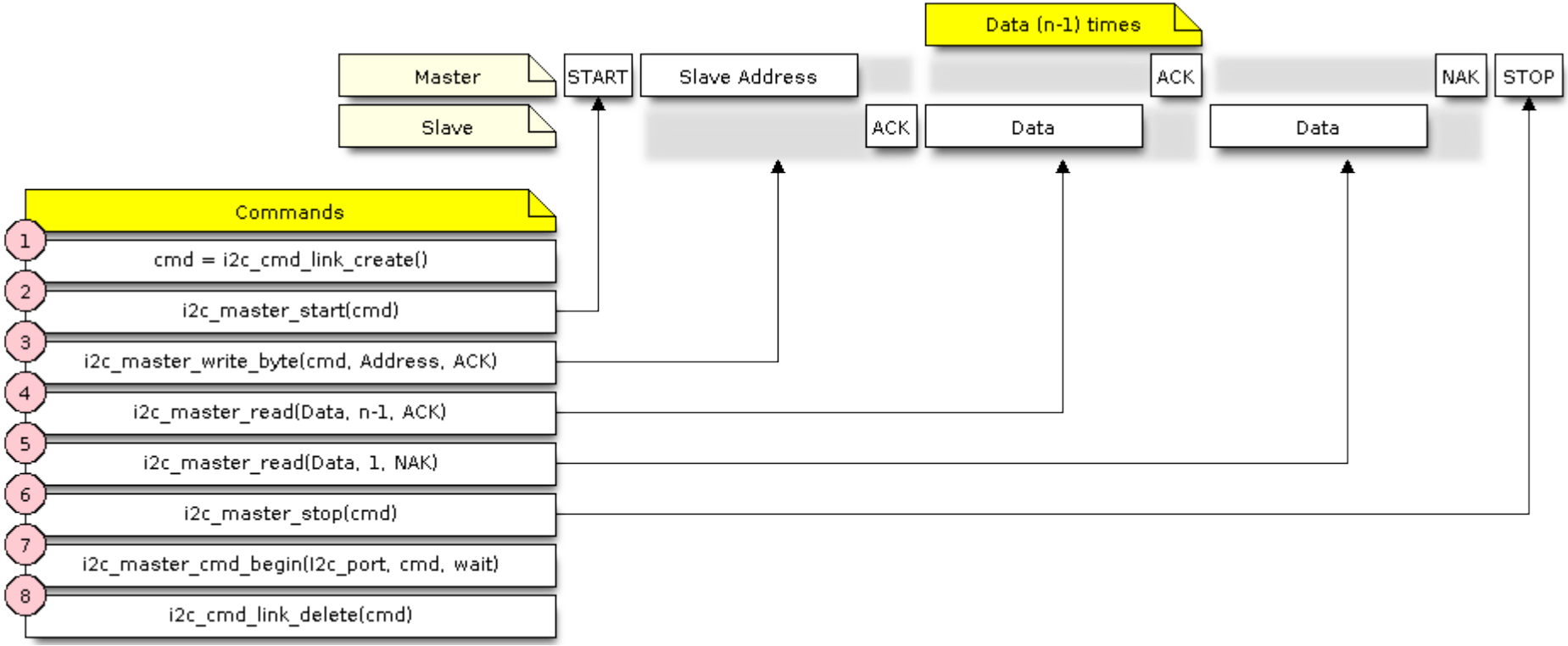


Last bit indicates Write or Read

Slave Acknowledges (or Not)

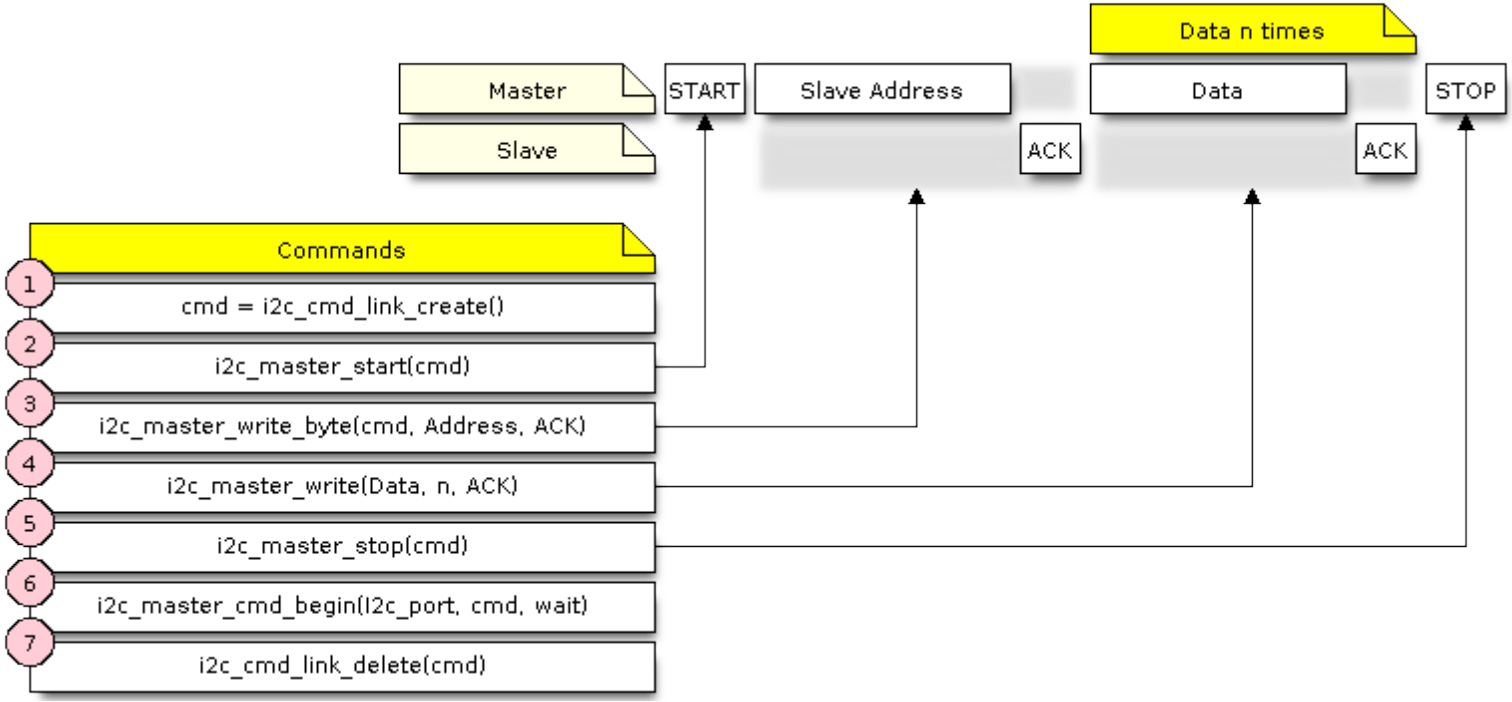
Master Asserts Stop condition

ESP32 Master Read Sequence



https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/i2c.html

ESP32 Master Write Sequence



https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/i2c.html

Master Writes to Slave
[normal Arduino – buggy on ESP32]

Master

```
#include <Wire.h>

void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop() {
  Wire.beginTransmission(8); // transmit to device #8
  Wire.write("x is ");       // sends five bytes
  Wire.write(x);             // sends one byte
  Wire.endTransmission();    // stop transmitting

  x++;
  delay(500);
}
```

https://www.arduino.cc/en/Tutorial/MasterWriter

Slave

```
#include <Wire.h>

void setup() {
  Wire.begin(8);             // join i2c bus with address #8
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600);        // start serial for output
}

void loop() {
  delay(100);
}

// function that executes when data is received from master
void receiveEvent(int howMany) {
  while (1 < Wire.available()) { // loop through all but the last
    char c = Wire.read(); // receive byte as a character
    Serial.print(c);      // print the character
  }
  int x = Wire.read();    // receive byte as an integer
  Serial.println(x);      // print the integer
}
```

Master Reads from Slave
[normal Arduino – buggy on ESP32]

Master

```
#include <Wire.h>

void setup() {
  Wire.begin();       // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}

void loop() {
  Wire.requestFrom(8, 6);    // request 6 bytes from slave  #8

  while (Wire.available()) { // slave may send less
    char c = Wire.read();   // receive a byte as character
    Serial.print(c);         // print the character
  }

  delay(500);
}
```

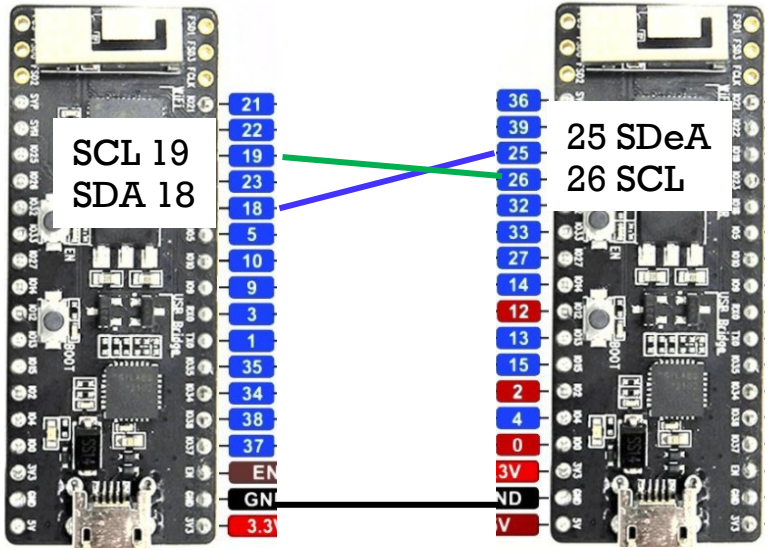
Slave

```
#include <Wire.h>

void setup() {
  Wire.begin(8);              // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}

void loop() {
  delay(100);
}

// function that executes when requested by master
void requestEvent() {
  Wire.write("hello "); // respond with message of 6 bytes
  // as expected by master
}
```

https://www.arduino.cc/en/Tutorial/MasterReader

ESP32 I2C Demo –on canvas [Not Arduino Wire Library]

1. Read slave buffer
   Print result
2. Delay 1 sec
3. Write slave
   Print sent data
4. Delay 1 sec
5. repeat



Master Slave

1. Read master
2. If something sent
   1. Print result
   2. Write to master
      Print sent data
3. repeat

I2C ESP32 to ESP32 Master

```
... // lots of init stuff before this code
// Canvas->files->resources->ESP32 Arduino Lecture Examples / i2c_master_lecture.ino
#define I2C_MASTER_SCL_IO (gpio_num_t)19
#define I2C_MASTER_SDA_IO (gpio_num_t)18

uint8_t data_wr[] = "GO Test     ";
uint8_t data_rd[DATA_LENGTH];

void setup() {
  Serial.begin(115200);
  i2c_master_init();
}

void loop() {
  if (i2c_master_read_slave(I2C_NUM_1, data_rd, DATA_LENGTH) == ESP_OK)
    Serial.printf("Read: %s\n",data_rd);
  delay(1000);
  if (i2c_master_write_slave(I2C_NUM_1, data_wr, DATA_LENGTH) == ESP_OK)
    Serial.printf ("Write: %s\n",data_wr);
  delay(1000);
}
```

I2C ESP32 to ESP32 Slave

```
... // lots of init stuff before this code
// Canvas->files->resources->ESP32 Arduino Lecture Examples / i2c_slave_lecture.ino
#define I2C_SLAVE_SCL_IO (gpio_num_t)26
#define I2C_SLAVE_SDA_IO (gpio_num_t)25

uint8_t data[] = "Message from slave     ";
uint8_t data_rd[DATA_LENGTH];

void setup() {
  Serial.begin(115200);
  i2c_slave_init();
}

void loop() {
  if (i2c_slave_read_buffer(I2C_NUM_0, data_rd, RW_TEST_LENGTH, 0) > 0 ) {
    Serial.printf("READ: %s\n",data_rd);
    if (i2c_slave_write_buffer(I2C_NUM_0, data, 10, 10/portTICK_RATE_MS) !=0 )
      Serial.printf("WRITE: %s\n",data);
  } // otherwise no data to read
  delay(10); // pretend we're doing something else here.
}
```

ESP32 IDF (I2C support)

Maximum wait time in ticks

i2c_slave_read_buffer(i2c_port_t n, uint8_t* data, size s, TickType_t t)

i2c_slave_write_buffer(i2c_port_t n, uint8_t* data, size s, TickType_t t)

i2c_master_write_slave(i2c_port_t n, uint8_t* data, size s)

i2c_master_read_slave(i2c_port_t n, uint8_t* data, size s)

I2C_NUM_0 or I2C_NUM_1

Pointer to Array of Data bytes data[]

integer number Of data bytes

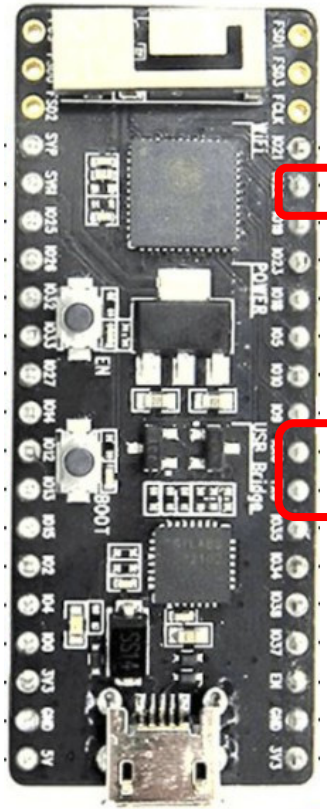
04

Other Protocols (I^2S, CAN, USB)

I2S (Inter-IC Sound)

- Closer to SPI than I2C
- Three wire (plus ground) interface
  - Clock (SCK) also called bitclock (BCLK or BCK)
  - Data (SD) also called (DATA, SDATA, SDIN, SDOUT…)
  - Word clock (WS) also called left-right clock (LRCLK), signals left stereo channel data and right stereo channel data
- Faster than I2C (up to 12Mbit/sec)
- Made for digital audio files (PCM), but can be used to transmit data.
- Standard hardware interface uses TTL voltage levels (5V)

I2S on Arduino on ESP32

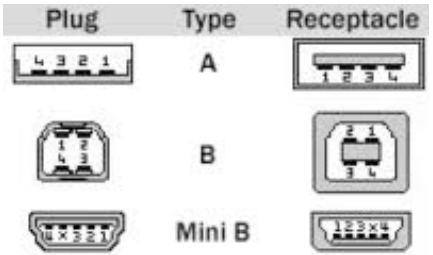


- I2S ESP32 IDF library
- For sound generation
- Probably will want to use SPIFFS for storing WAV files.
- Can also directly output to DAC channels for analog sound generation

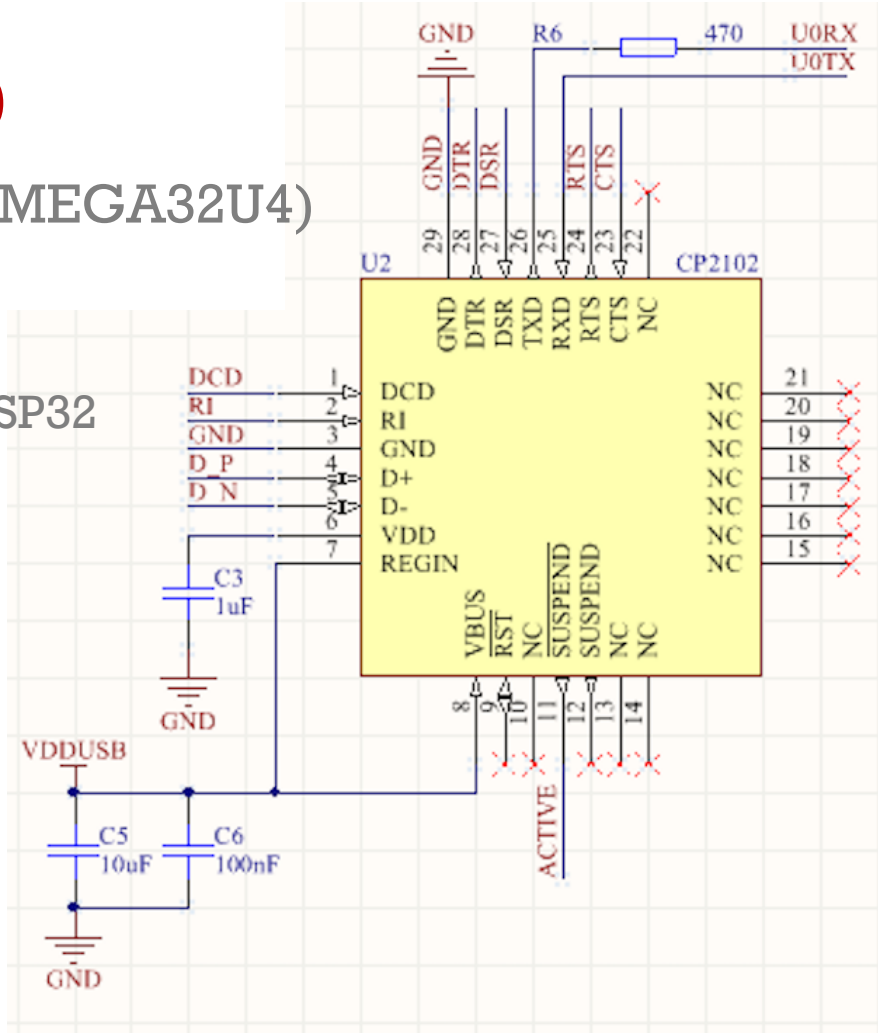
https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/i2s.html

USB (Universal Serial Bus)

- Either USB included on chip (ex: ATMEGA32U4) or
- Use USB driver bridge
  - to UART (U0TX, U0RX) ex: CP2102 on ESP32
  - to SPI ex: CP2130

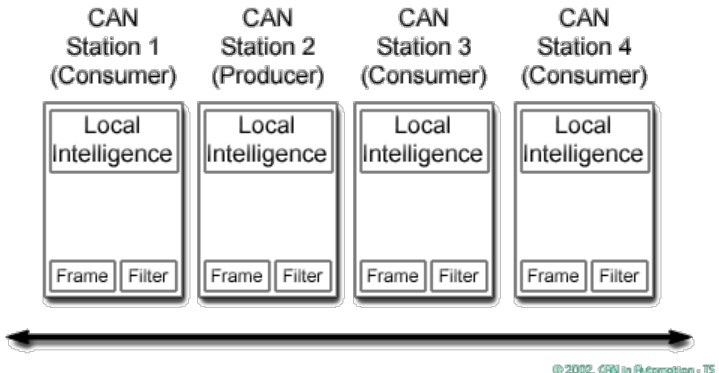


Pin	Signal	Color	Description
1	VCC		+5V
2	D-		Data -
3	D+		Data +
4	GND		Ground



CAN Bus (Controller Area Network)

- Layer 2 (data link layer) of OSI model
  - Packet transmission protocol
  - Collision, error detection (all in hardware!)
- Uses differential pair (e.g., twisted pair)
  - Two lines relative to each other (not gnd)
  - Robust to electric noise (e.g., in a car)
- Up to 1 megabit/sec.



Comparing Serial Protocols

Estimated Parameters

	UART – RS232	UART- Logic level	I2C	SPI	CAN	USB 2.0
Network Architecture	Singlemaster-Singleslave	Singlemaster-Singleslave	Multimaster-Multislave	Singlemaster-Multislave	Multimaster-Multislave	Singlemaster-Singleslave
# signal lines (add gnd)	2+5 optional full-duplex	2 full-duplex	2 half-duplex	4 full-duplex	2 half-duplex	2 half-duplex
Size of network	15m @ 20kbs	~1m @ 20kbs	7-bit addr limited by 400pF	limited by SS lines	11-bit addr	1 to 1
Interface	+/- 12V logic level	logic level	Open drain active master	Push-pull and tristate	Differential pair	Differential pair
Voltage	min -5V and 12V required	typ 3.3V or 5V	1.8 to 5.5V	1.8 to 5V	min 4.5V diff	5V signal
Speed	1.5Mb	1.5Mb	~400kbp	~12Mb	~1Mbps	280Mbps

Other concerns

- FIFO buffer depth
  - When multi-tasking, MCU's need some buffer to hold data before processes can get to them. This can be critical for reliable data transfer on a busy MCU
- Line length – noise will become an issue
- Packetizing protocol
  - Often data sent won't be received, OSI-ISO level 2 handles this and resends if necessary.
- Debugging tools
  - Packet analyzer (sniffer) can be useful
  - Logic Analyzer and/or Oscilloscope

Summary Quiz

Q6 For asynchronous comm, you must set the same _____ on both ends

Q7 For synchronous, _____ sets the clock speed (mostly, though I2C has slightly different)

Q8 _____and ____ are two most common *synchronous* protocol used between embedded processors

Q9 _____ can be used for simple limited state changes