18-100 Introduction to Electrical and Computer Engineering

Lecture 14

Serial Communication

A Note on Terminology

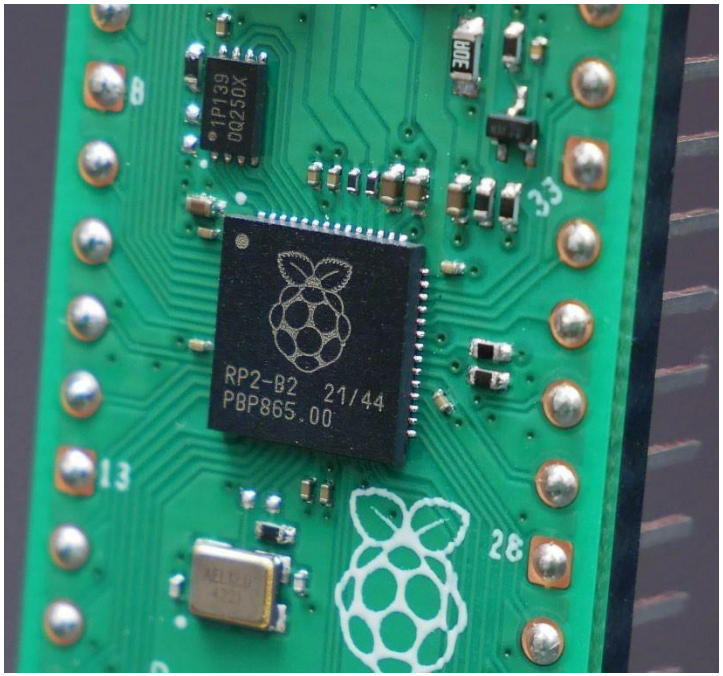
- Many specifications refer to the role of devices using now deprecated language
  - Masters and Slaves

- Although there was probably little reflection on this language years ago, that isn't the case today
  - We won't be using this language
  - When you become engineers and write specification, you shouldn't use this language
  - We mention only this language because it is presently unavoidable
    - It is written into many formal specifications.
    - You'll see it if you google many of today's lecture topics, etc.

- Use language that is both more descriptive and neutral, even if a formal spec does otherwise.
  - **Coordinator and Participant/Device**

Microcontrollers

- Microcontrollers are microprocessors with peripherals and memory
- **Peripherals:** Interfaces with the outside world
  - ADC (analog-to-digital converters)
  - Serial Interfaces
  - GPIO (General Purpose I/O)
- Often used in **embedded systems** – which do specialized computing to control a device or system



Raspberry Pi Pico

Serial Communication Protocols

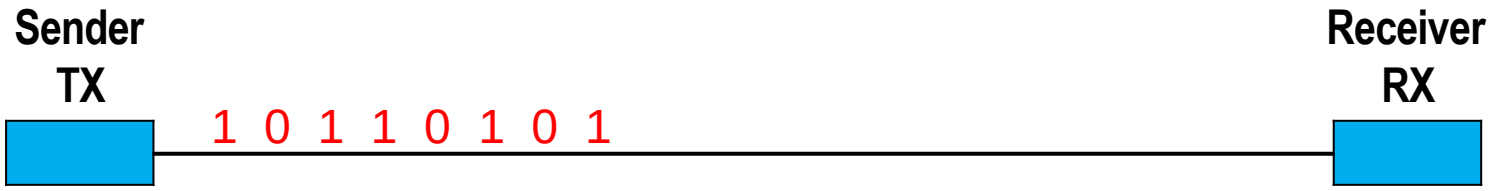
`0101011100010100010011111010010101111`

"The single biggest problem in communication is the illusion that it has taken place."

- George Bernard Shaw

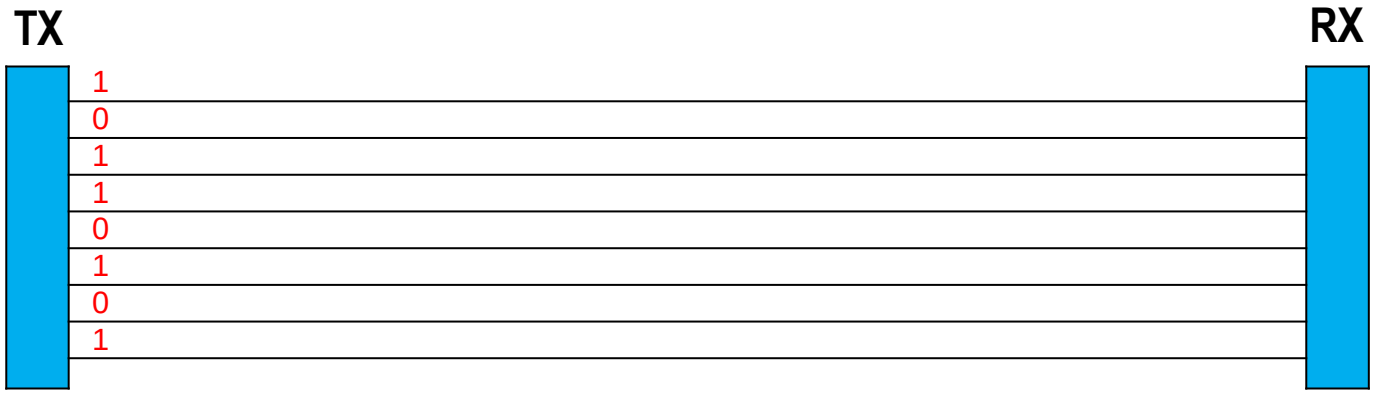
Serial vs. Parallel Communication

Serial sends bits sequentially



- Less wires, at lower speeds

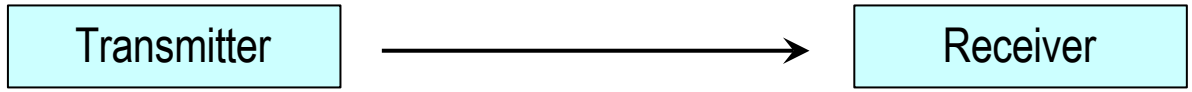
Parallel sends multiple bits simultaneously



- More wires, but higher speeds

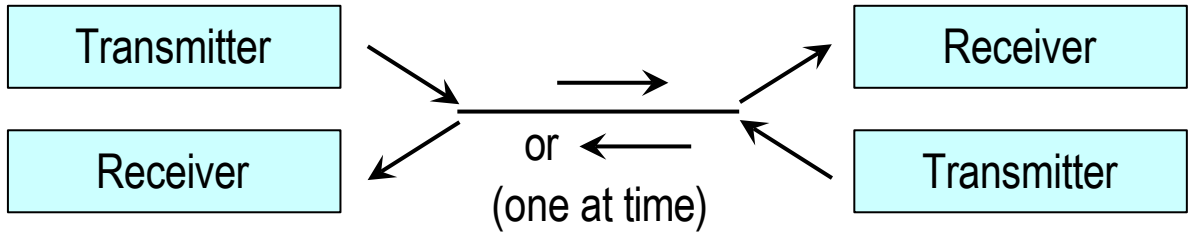
Simplex vs. Half vs. Full Duplex

Simplex



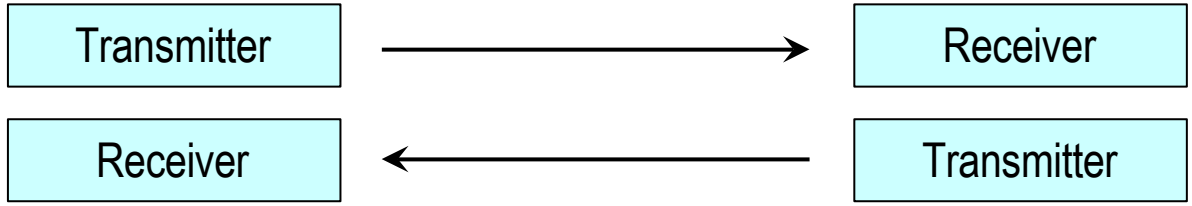
Examples: Monitor, Keyboard, LED Display, Printer, RFID, …

Half Duplex



Examples: Walkie-Talkie, I2C, SPI…

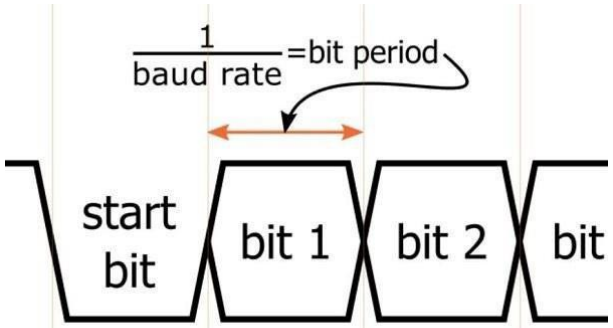
Full Duplex



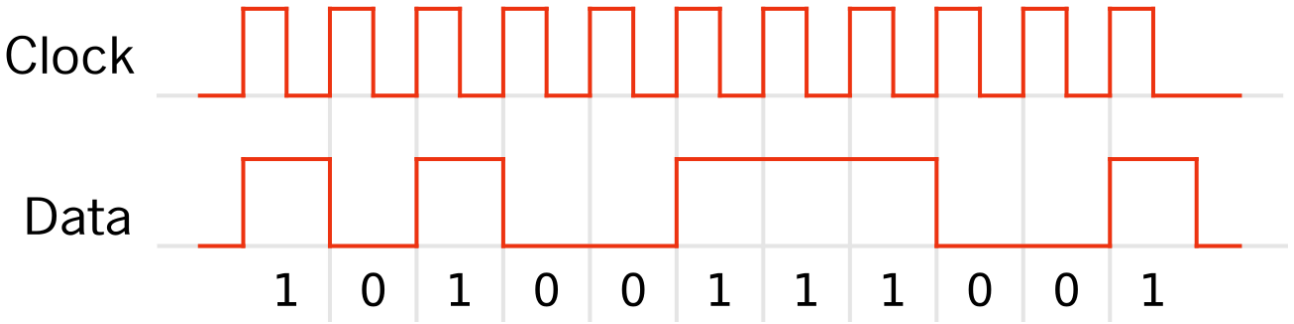
Examples: Telephone, Cell phone, UART

Terms and Definitions

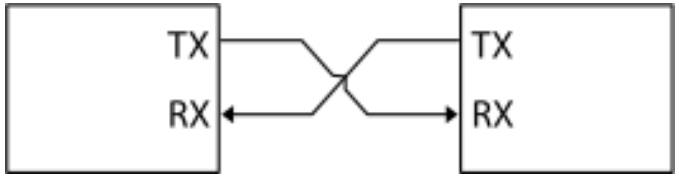
- **Data Rate:** How many 1's and 0's the device can transmit/receive (in bps, bits per second, or "baud")



- **Synchronous vs. Asynchronous:** Is the data being sent aligned to a clock signal?



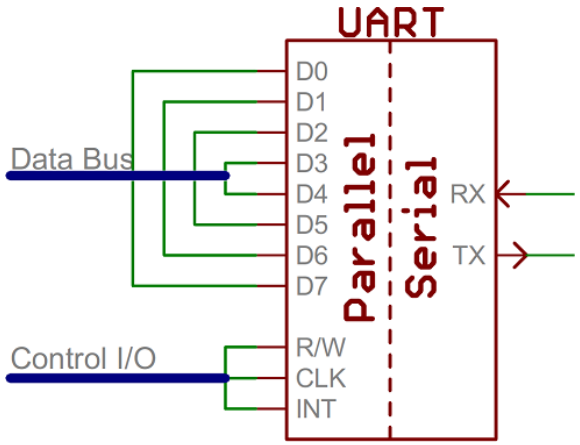
UART



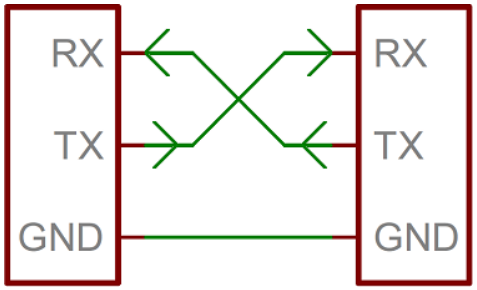
SPI

I2C

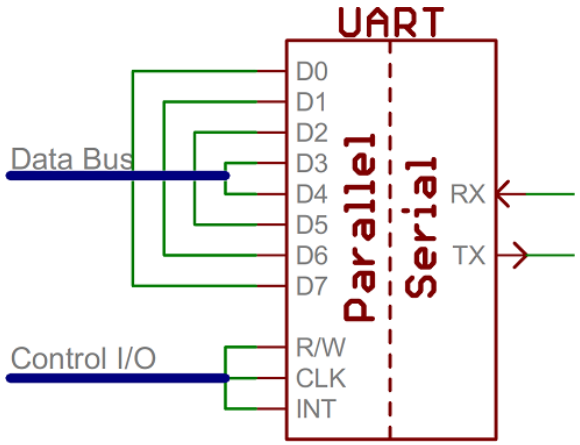
UART – Universal Asynchronous Receiver-Transmitter



- Most common digital communication peripheral device
  - Frames data (More soon)
  - Sets timing
  - Provides logical encoding
  - Physical encoding provided by line driver circuit, e.g. RS-232, RS-422, RS-485

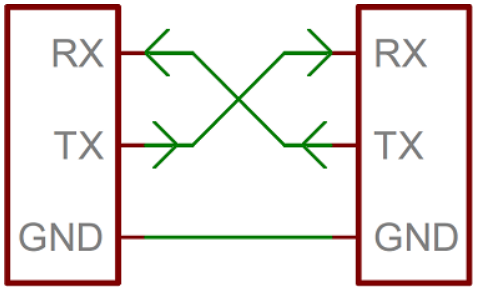


UART – Universal Asynchronous Receiver-Transmitter



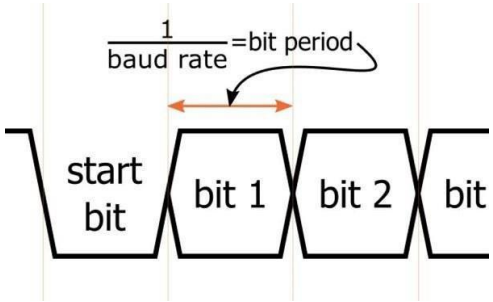
- Typically, 2 pins are used
  - RX – Receiver
  - TX – Transmitter

- UART mode can be half-duplex or full-duplex
  - Connect RX to TX and TX to RX for full-duplex
  - Connect only one pin for half-duplex
  - **RX/TX are from the perspective of the device!**



UART – Baud Rate

- Receiver and Transmitter Agree on a "baud rate" and encoding
  - How fast should the transmitter send information?



- Baud rate is the "symbol rate"
  - Simple models use one bit per symbol
  - More complex models may communicate more than one bit per symbol, e.g. multiple signal levels, phasings, etc.

- Typical baud rates include:
  - *300, 1200, 2400, 4800,* ***9600****, 19200, 38400, 57600,* ***115200***
  - Mismatches in between the sender and receiver lead to transmission issues

UART – Framing

- Framing is, perhaps the most important aspect of digital communication
- It breaks a stream of data into manageable units
- Each frame can then be individually transmitted, recognized, and received.
- Each frame can then also potentially be checked for integrity and possibly have other added value, e.g. addressing
- Without framing, there would be no units of data to manage, at best it could just flow as is

UART – How to send a message

- Framing is, perhaps the most important aspect of digital communication
- It breaks a stream of data into manageable units

- Typical UART "**frame**":

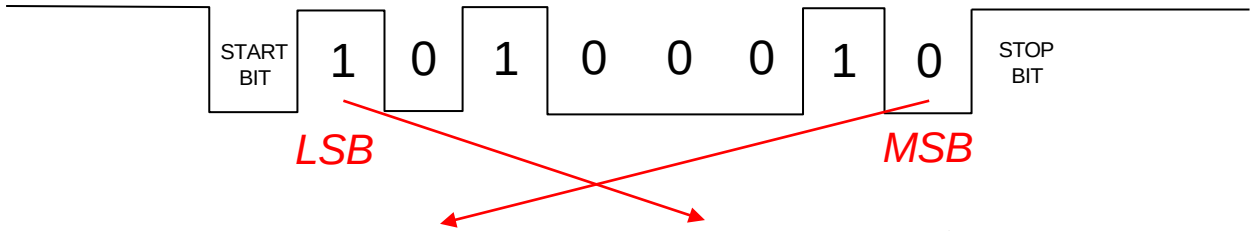
Start Bit (1 bit)	**Data Frame** (5 to 9 bits)	**Parity Bits** (0 to 1 bits)	**Stop Bits** (1 to 2 bits)

- 8-N-1 is the most common configuration
  - (8) bits of data, (N)o Parity Bit (E = even, O = odd, N = none), (1) Stop Bit
- Example:

Start Bit (1 bit)	**Data Frame** (8 bits)	**Stop Bit** (1 bit)

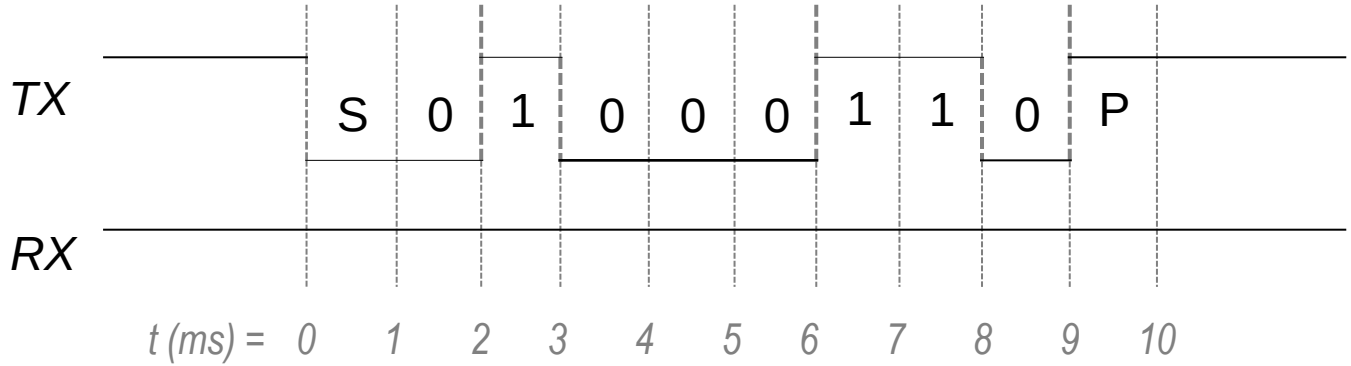
UART – Reading a UART Message

- For serial data, LSB comes first!



61	110 0001	a
62	110 0010	b
63	110 0011	c

- Example:



Sending Text: ASCII Encoding

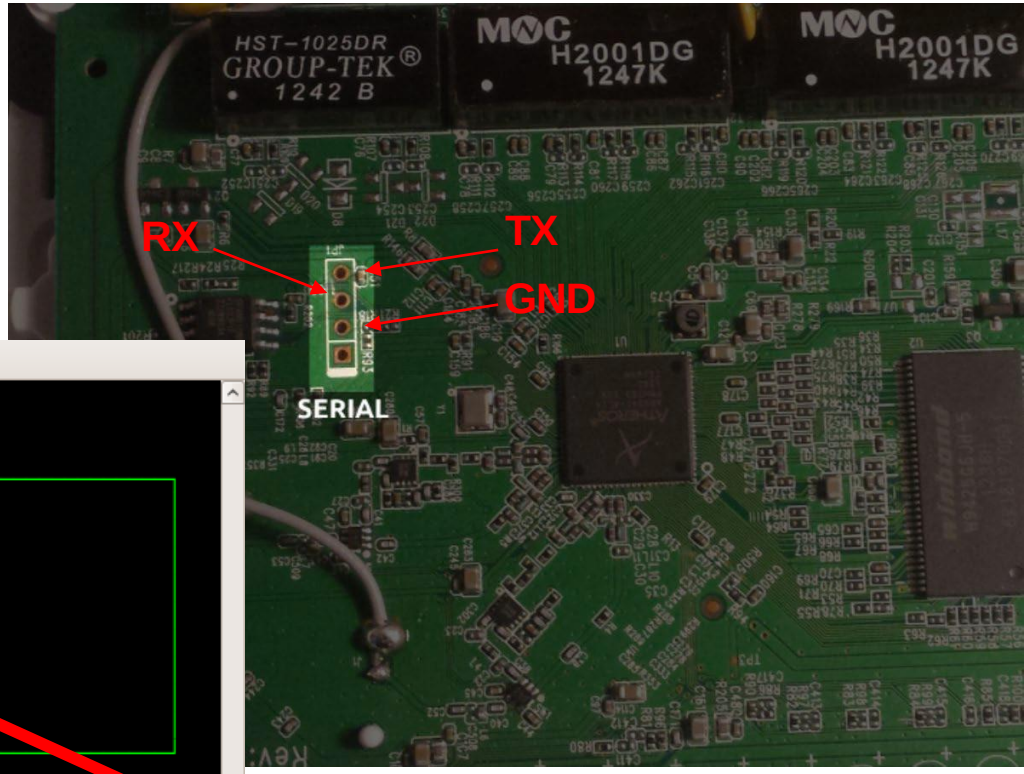
- ASCII is **character encoding** scheme that uses 7 bits

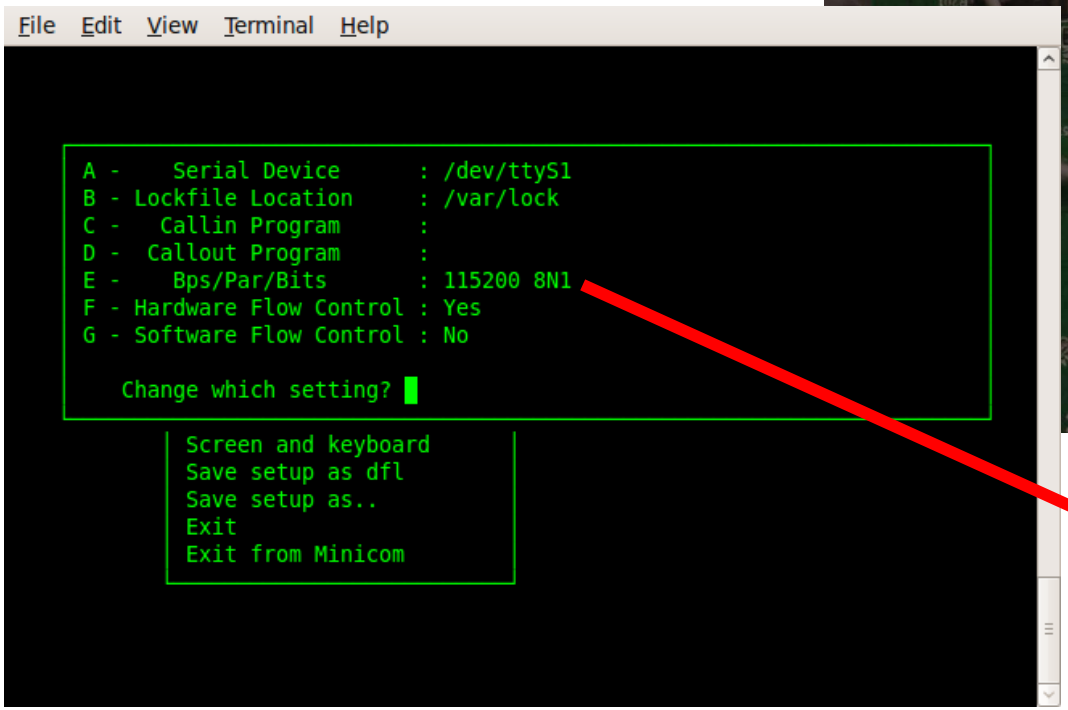
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

- Example: 0x45 = `1000101`=`'E'`, 0x52 = `1010010`=`'R'`

UART – In the wild

- Most Consumer Electronics/IoT Devices have a UART port for debugging purposes





115200 Baud
1 byte data
no parity
1 stop bit

UART – Summary

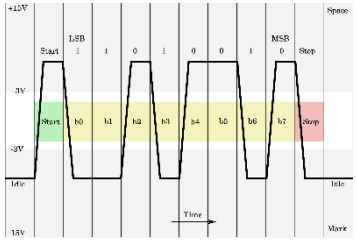
- UART is a type of hardware device, not a protocol specification
  - Lower level protocols such as RS-232, RS-422, RS-485, etc, specify the signaling
  - 2 pins RX (receiver), TX (transmitter)
  - Asynchronous (however a synchronous variant, USART, exists)
  - Half or Full-Duplex (depending on if both RX/TX are used)
  - Typical frame (8N1):

Start Bit (1 bit)	Data Frame (8 bits)	Stop Bit (1 bit)

- Advantages
  - Simple to implement; only needs two wires for full-duplex, no clock needed
  - Extremely common (almost all microcontrollers have at least one UART)
- Disadvantages
  - Requires both devices to agree on a Baudrate and frame structure
  - Typically used for relatively slow and point-to-point communication

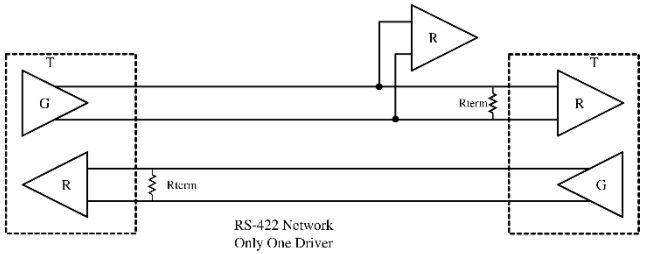
RS-232 vs RS-422 vs RS-485

- RS-232 is a simple line-level protocol managed by UARTs.



https://en.wikipedia.org/wiki/RS-232#/media/File:Rs232_oscilloscope_trace.svg

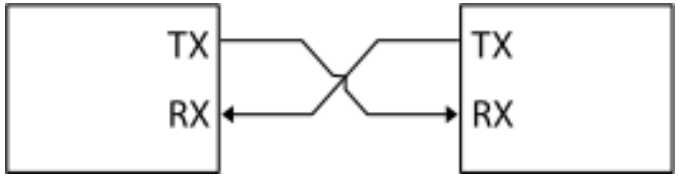
- RS-422 is "improved" in the sense that it uses differential signaling, driving both lines, rather than just one line, removing the common ground reference, and improving range and speed. It also allows multiple receivers



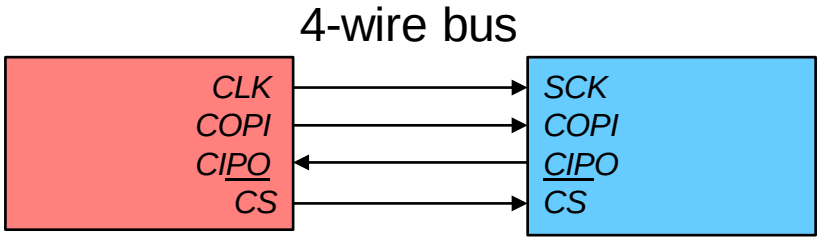
https://en.wikipedia.org/wiki/RS-422#/media/File:RS-422_Network.svg

- RS-485 uses signaling like RS-422, but has "tri-state" driver circuits such that individual stations can release the line, enabling more than one station to be able to send.

UART



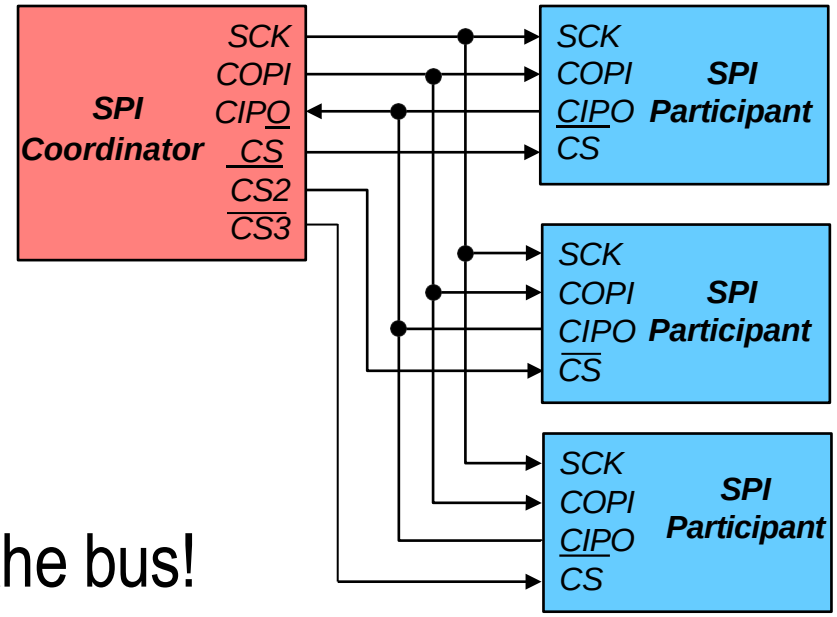
SPI



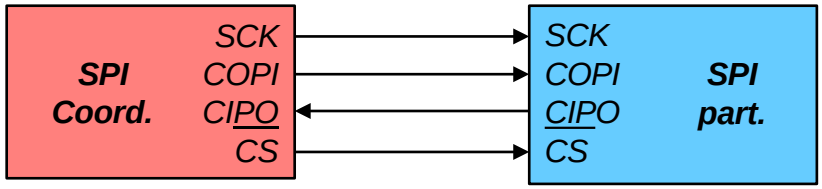
I2C

SPI – Serial Peripheral Interface

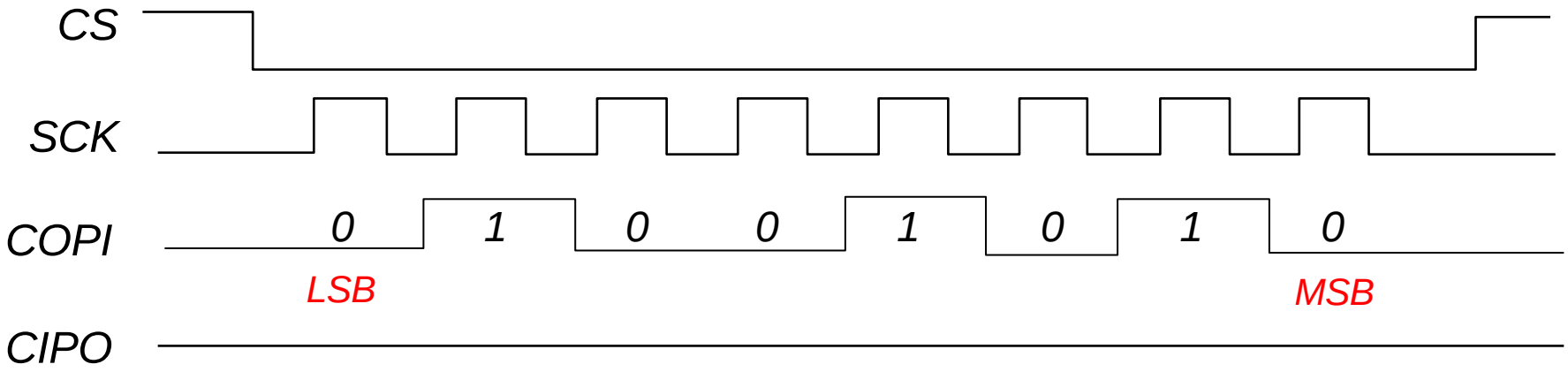
- Synchronous Serial Protocol
- "4 + N-wire" serial bus
  - SCK – Clock
  - COPI – Coordinator out, Participant in
  - CIPO – Coordinator in, Participant out
  - $\overline{CS}$ – Chip Select, one per participant
- Can have multiple devices on the bus!
  - Only 1 coordinator device
  - Multiple participant devices (one per CS pin!)
- Coordinator controls the clock signals and the chip select(s)



SPI – Sending a SPI Message



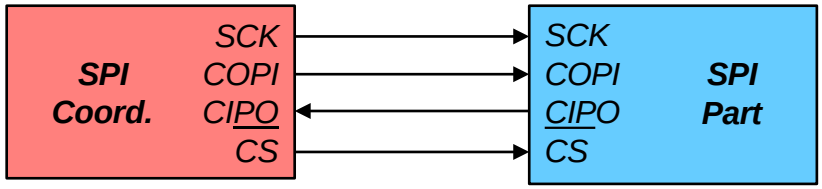
Send a message from coordinator to participant:



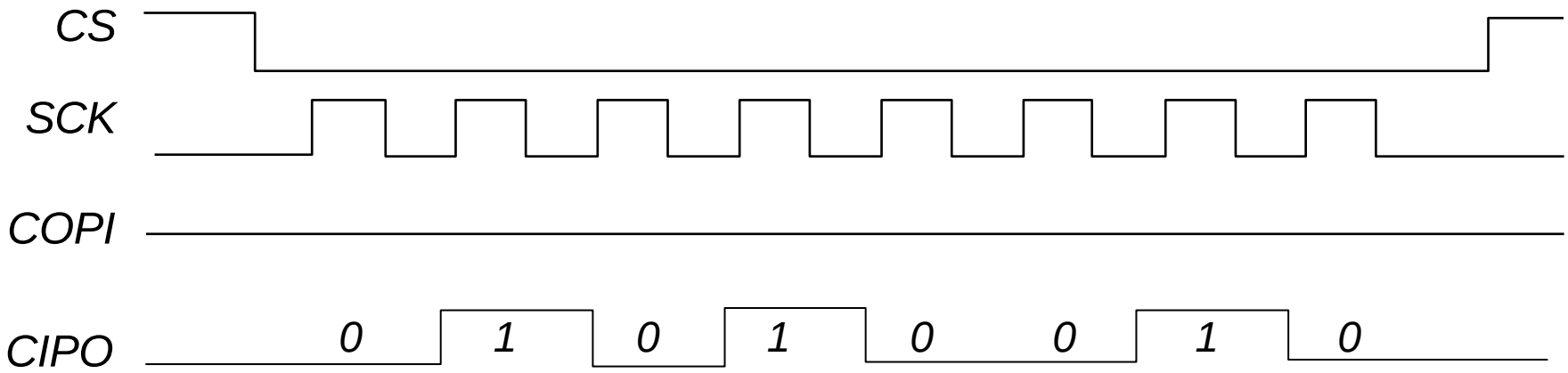
Examples:

`'R' = 0x52 = 01010010` `'J' = 0x4A = 01001010`

SPI – Sending a SPI Message



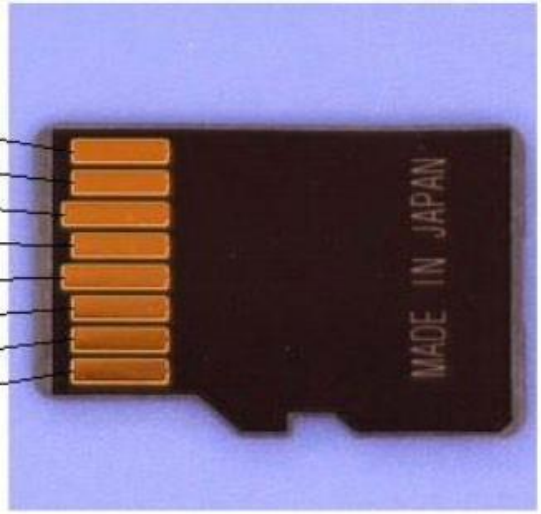
Send a message from coordinator to participant:

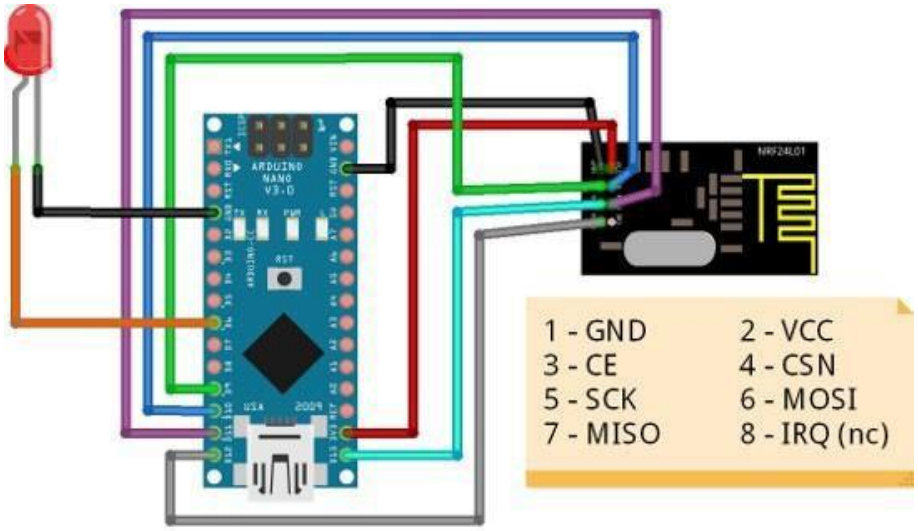


SPI – In the wild

microSD

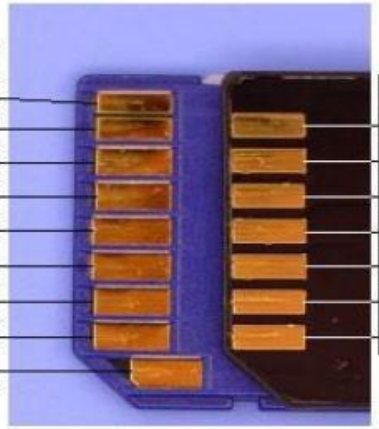
No	SD	SPI
8	DAT1	
7	DAT0	DO
6	Vss	
5	CLK	SCLK
4	Vdd	
3	CMD	DI
2	DAT3	CS
1	DAT2	





SDC

No	SD	SPI
8	DAT1	
7	DAT0	DO
6	Vss2	
5	CLK	SCLK
4	Vcc	
3	Vss1	
2	CMD	DI
1	CAT3	CS
9	DAT2	

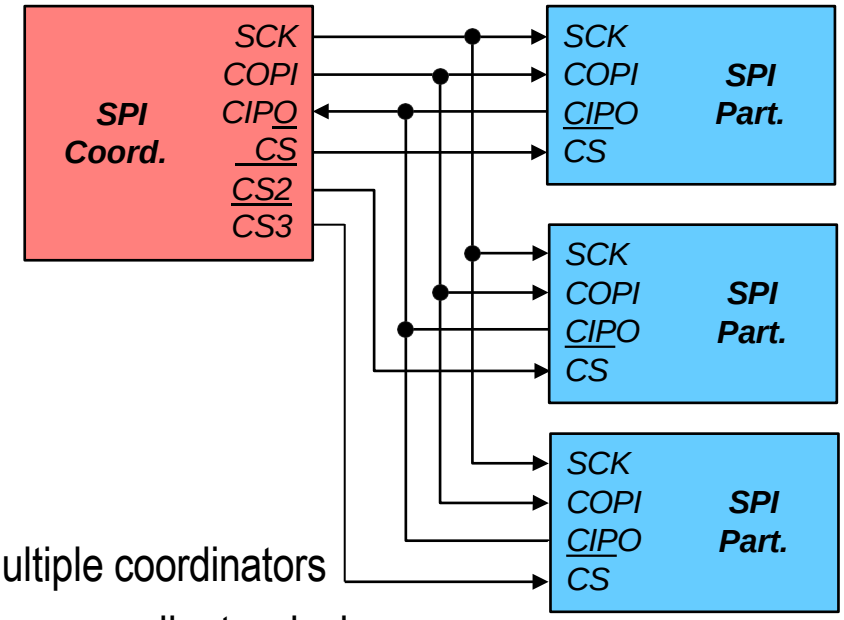


MMC

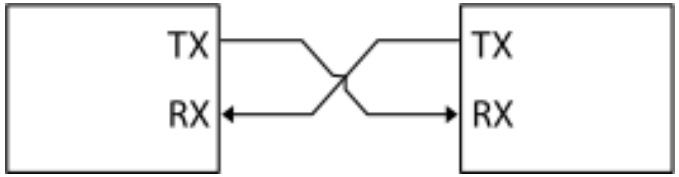
No	MMC	SPI
7	DAT	DO
6	Vss2	
5	CLK	SCLK
4	Vcc	
3	Vss1	
2	CMD	DI
1	RES	CS

SPI – Summary

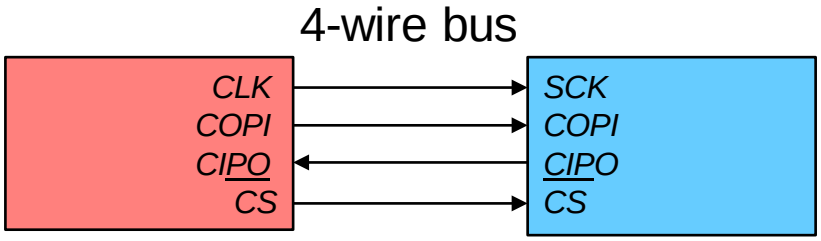
- "4-wire" serial protocol
  - SCK (clock), COPI (coord. out, part. in), COPI (coord. out, par. in), CS (chip select)
  - Synchronous (has a common clock)
  - Full-Duplex
- Advantages
  - Very fast (up to a few MHz!!)
  - Can have multiple devices on a bus, even multiple coordinators
  - Simple hardware interface: only need 4 pins per coordinator device
- Disadvantages
  - Need a new wire per additional participant device ($$$)
  - Participant devices can't acknowledge messages or adjust timing
  - Limited to short distances (<10m ~ 30ft)
  - No actual standard to work from (pins may even have different names!)
    - SCK = CLK, MOSI = SDI = CIPO, MISO = SDO = COPI, CS = SS = CE = PS



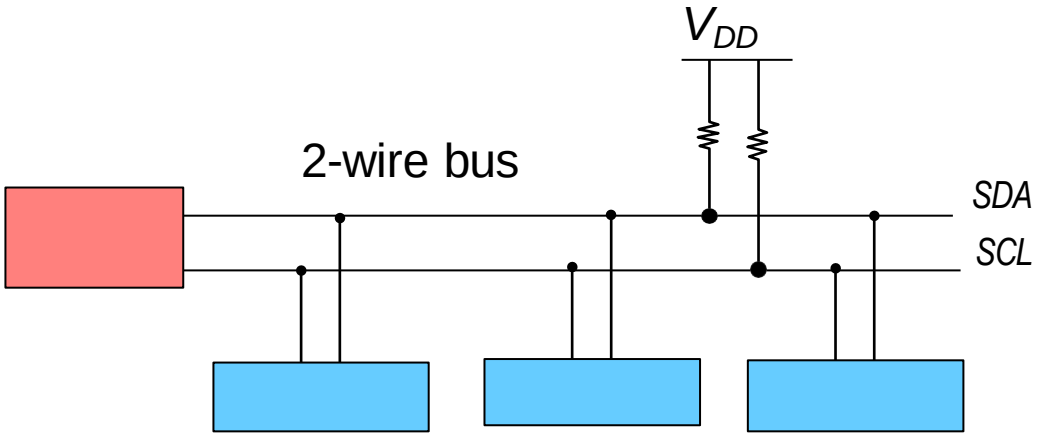
UART


TX
RX
TX
RX

SPI

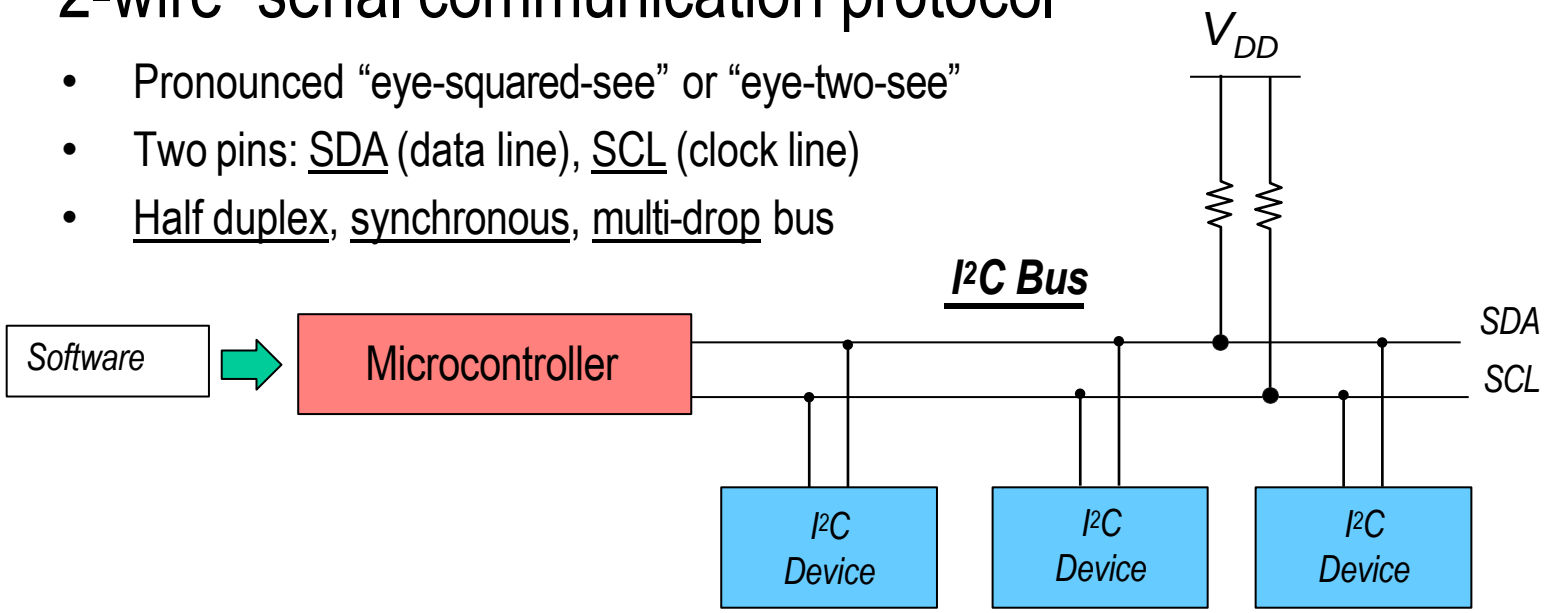

4-wire bus
CLK
COPI
CIPO
CS
SCK
COPI
CIPO
CS

I2C


V_{DD}
2-wire bus
SDA
SCL

I²C – Inter-Integrated Circuit

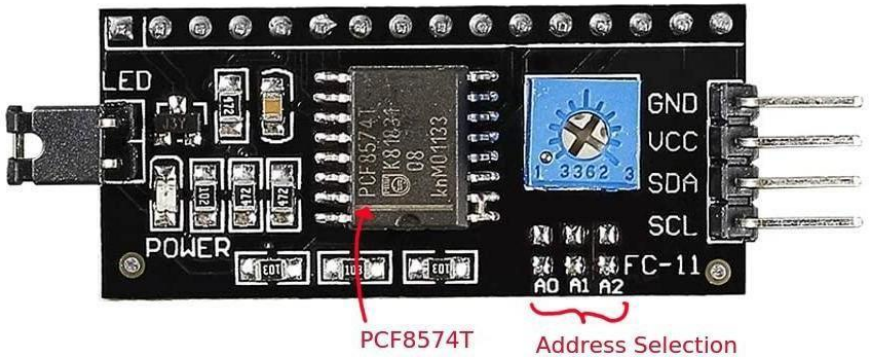
- "2-wire" serial communication protocol
  - Pronounced "eye-squared-see" or "eye-two-see"
  - Two pins: SDA (data line), SCL (clock line)
  - Half duplex, synchronous, multi-drop bus



- By default, the line voltage is HIGH
  - Devices required to pull lines LOW
  - Uses pull-up resistors (typically integrated into microcontrollers)

Device Addresses

- Each device connected to the bus has a unique identifier: **a 7-bit address**
- Most devices have some address bits hard-coded, while a few of the LSBs are reconfigurable, allowing multiple devices on the same bus.



LCD Serial to Parallel Converter

Address: 0x27

A6	A5	A4	A3	A2	A1	A0
0	1	0	0	1	1	1

- The coordinator uses this address to call out the device it would like to interact with

I²C – Sending a message

- Two types of frames:
  - Address frames (7-bit address, 1-bit read/write)

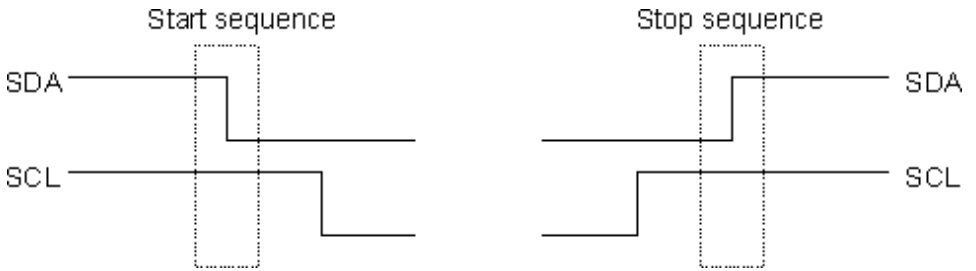
  - Data frames (8 bits of data)

- Each device on the bus gets a unique 7-bit address
  - Every device gets every message; devices are looking for their address
- Coordinator device indicates whether its reading or writing with the R/W bit
  - Write = LOW, Read = HIGH
- Device indicates in received a message with an acknowledge
  - ACK -> SDA pulled LOW by device, NACK -> SDA **not** LOW (default HIGH)

I²C – Typical Exchange

S	Address Frame (7 bits)	R/W (1 bit)	ACK (1 bit)	Data Frame (8 bits)	ACK (1 bit)	Data Frame (8 bits)	ACK (1 bit)	P

- START/STOP conditions when SDA then SCL goes LOW

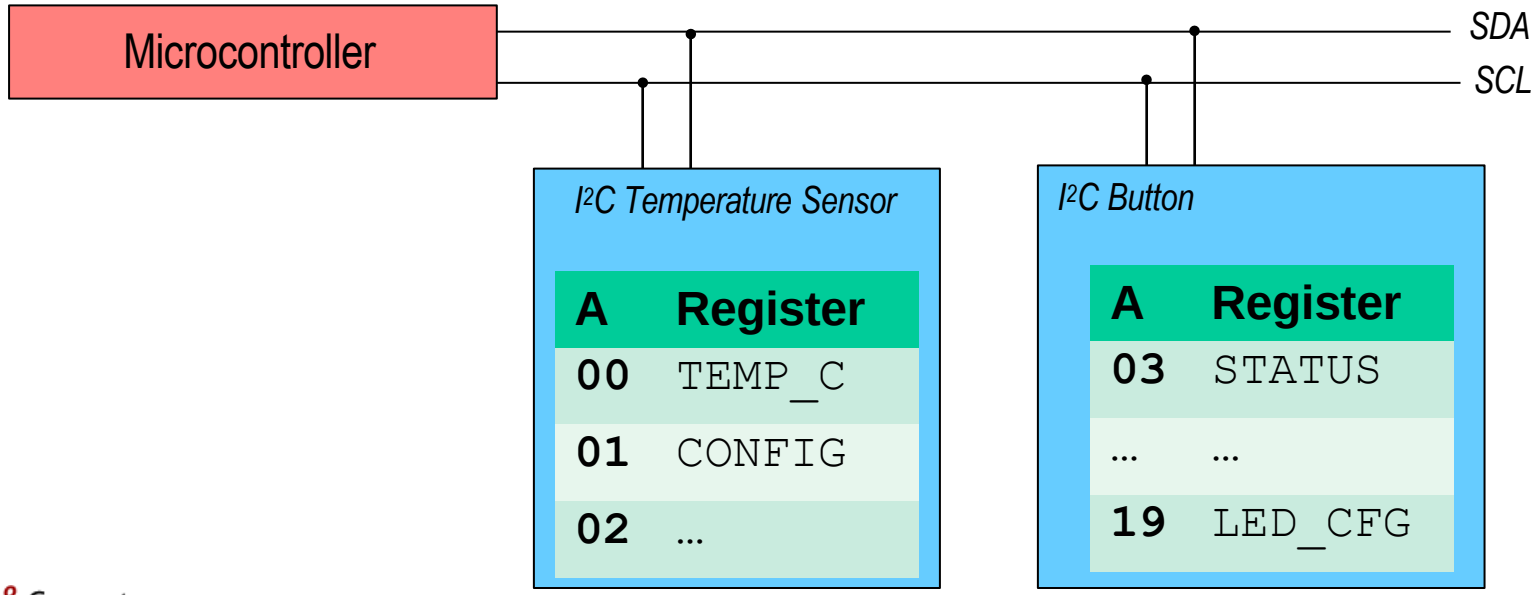


SDA

SCL

I²C – Device Registers

- Three main types of registers
  - **Data Registers:** Data to be accessed on the bus
  - **Status Registers:** Contain information about the device's operation (read only)
  - **Config Registers:** Used to change aspects of the device (read/write)
- Every register has an address
  - This is different from the device's I²C address



I²C – Device Registers

- Temperature Sensor register (16-bit)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEMP[11:0]												Reserved			
r	r	r	r	r	r	r	r	r	r	r	r				

TEMP = raw temperature value, r = read only, Reserved = not used.

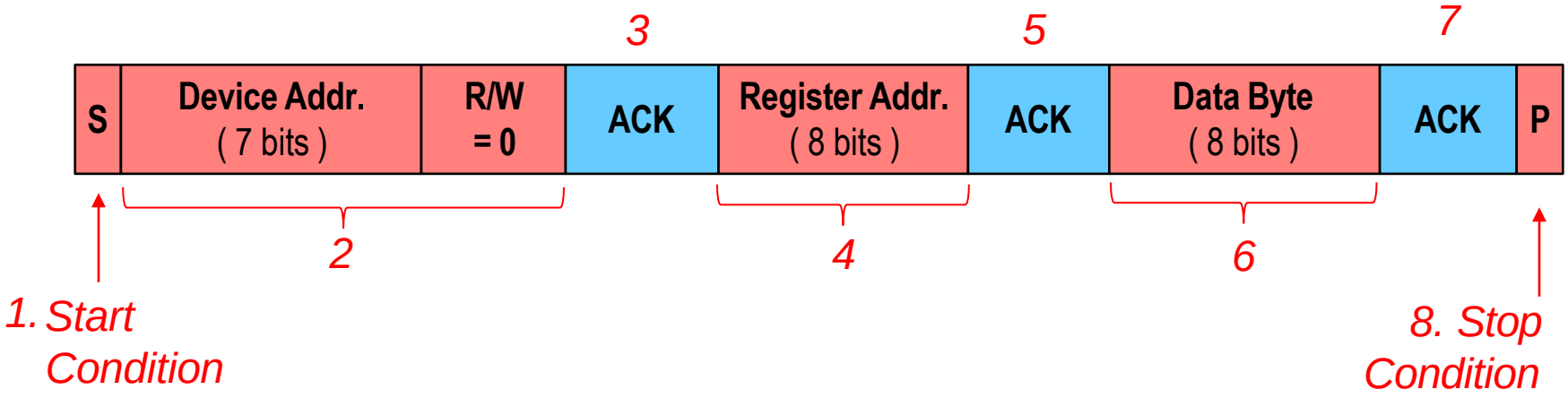
Figure 5: TMP102 Register Layout

- Button Registers (8-bit, 16 registers)

```
ID = 0x00,
FIRMWARE_MINOR = 0x01,
FIRMWARE_MAJOR = 0x02,
BUTTON_STATUS = 0x03,
INTERRUPT_CONFIG = 0x04,
BUTTON_DEBOUNCE_TIME = 0x05,
PRESSED_QUEUE_STATUS = 0x07,
PRESSED_QUEUE_FRONT = 0x08,
PRESSED_QUEUE_BACK = 0x0C,
CLICKED_QUEUE_STATUS = 0x10,
CLICKED_QUEUE_FRONT = 0x11,
CLICKED_QUEUE_BACK = 0x15,
LED_BRIGHTNESS = 0x19,
LED_PULSE_GRANULARITY = 0x1A,
LED_PULSE_CYCLE_TIME = 0x1B,
LED_PULSE_OFF_TIME = 0x1D,
I2C_ADDRESS = 0x1F
```

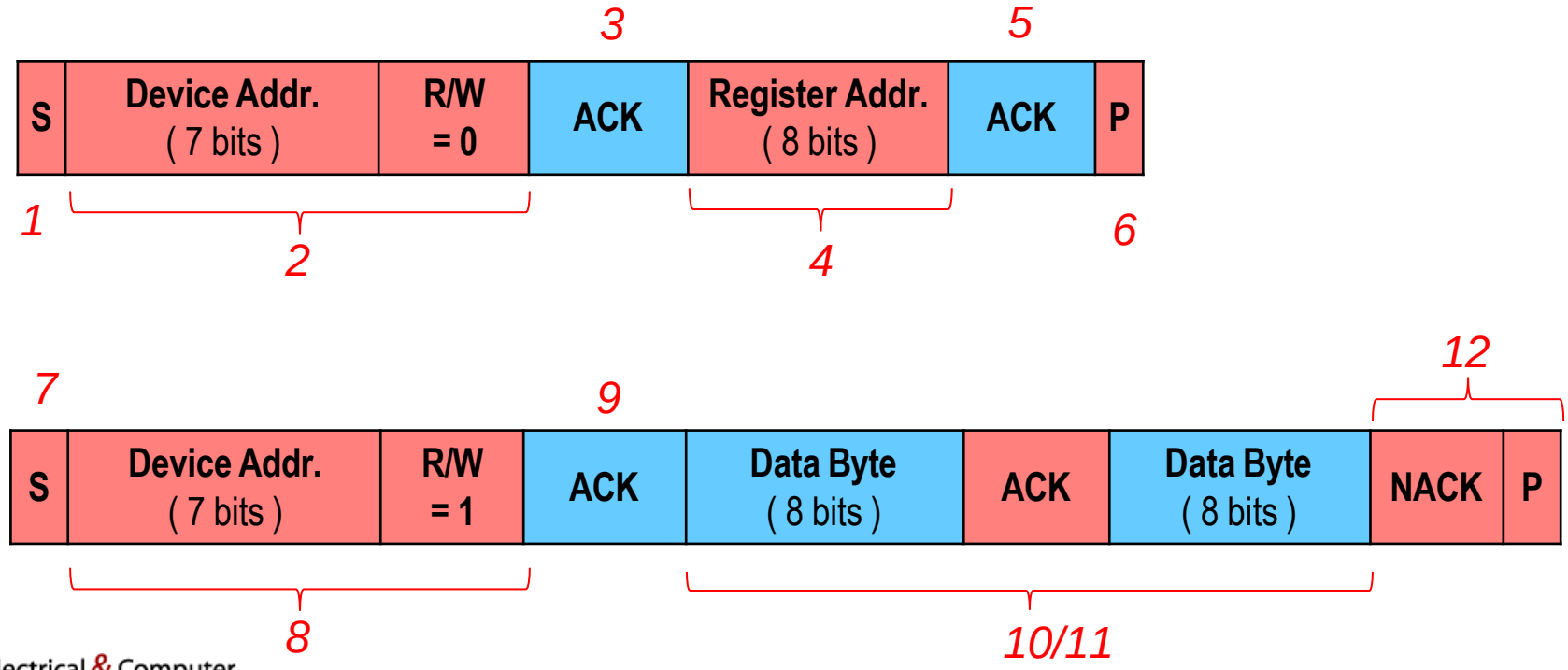
I²C – How to write to a register

1. Send **Start condition**
2. Send participant address (7 bits) + **Write** bit (0)
3. Wait for participant to **ACK** (pull data line LOW)
4. Send **register address** in data frame
5. Wait for **ACK**
6. **Send data** byte(s)
7. Wait for **ACK**
8. Send **Stop condition**

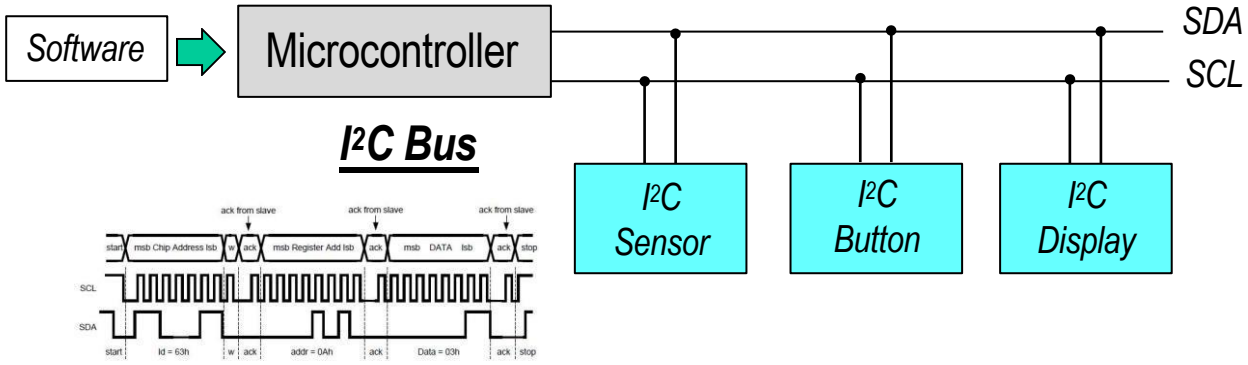


I²C – Read from a register

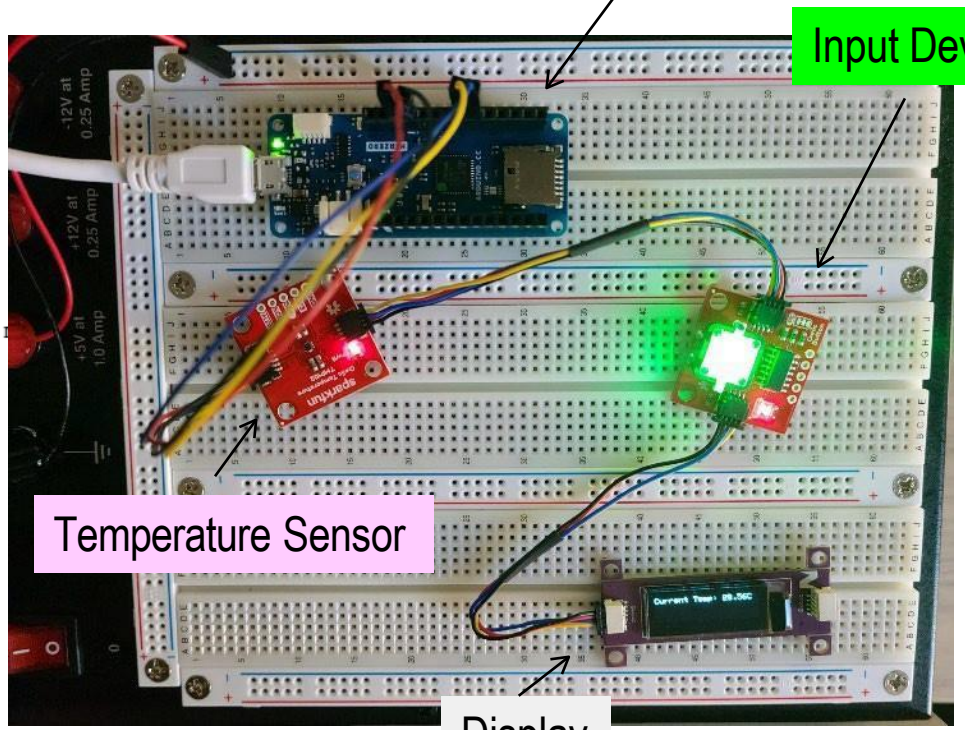
1. Send **Start condition**
2. Send part. address (7 bits) + **Write** bit (0)
3. Wait for part. to **ACK**
4. Send **register address** in data frame
5. Wait for **ACK**
6. Send **Stop condition**
7. Send Repeated **Start condition**
8. Send part. address (7 bits) + **Read** bit (1)
9. Wait for **ACK**
10. Receive **Data** byte(s)
11. Coordinator sends **ACK**
12. Send **NACK** and **Stop** after last byte



I²C Example




```
10    sensor.begin();
11    delay(100);
12  }
13
14  void loop()
15  {
16    // Turn sensor on to start temperature
17    sensor.wakeup();
18
19    // read temperature data
20    temperature = sensor.readTempC();
21
22    // Print temperature
23    Serial.print("Temperature: ");
24    Serial.print(temperature);
25    Serial.println();
26
27    delay(300);  // Wait 0.3s
28  }
```

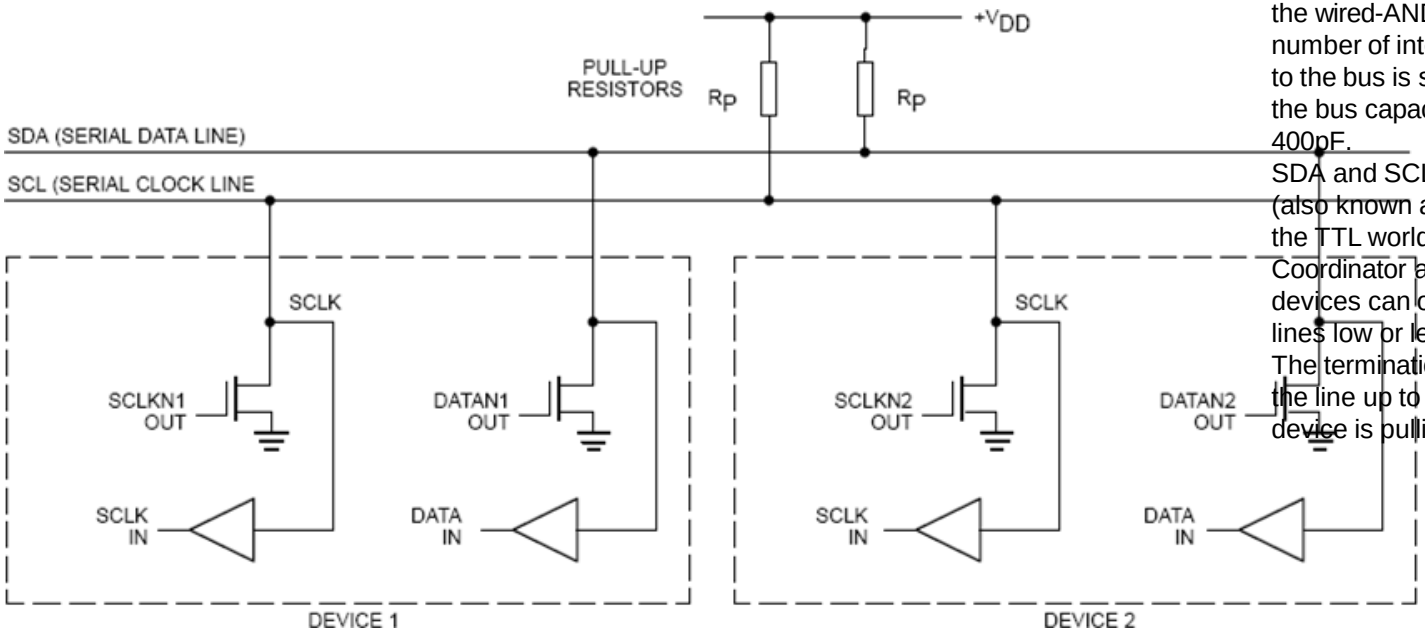


Presenter Notes
2023-04-10 18:50:51

Both SDA and SCL are bidirectional lines, connected to a positive supply voltage via a pull-up resistor (see Figure 4). When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain or open-collector in order to perform the wired-AND function. The number of interfaces connected to the bus is solely dependent on the bus capacitance limit of 400pF.

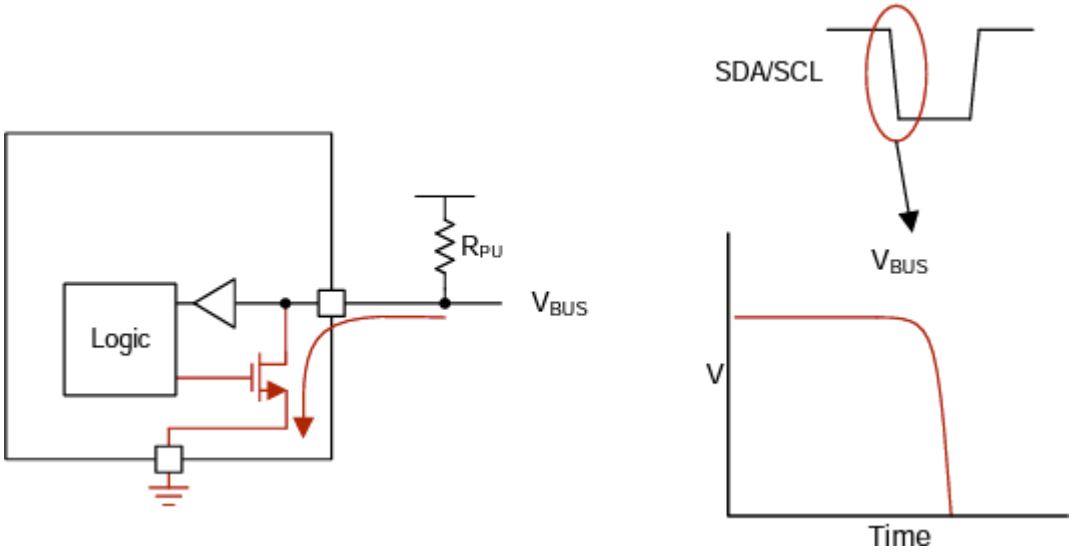
SDA and SCL are open-drain (also known as open-collector in the TTL world), that is I2C Coordinator and Participant devices can only drive these lines low or leave them open. The termination resistor Rp pulls the line up to Vcc if no I2C device is pulling it down.

Connecting I2C Devices to the Bus



- Devices read the bus voltage through a buffer (SCLK IN, DATA IN)
- Devices pull down the bus voltage through a drain-connected MOSFET (SCLK OUT, DATA OUT)

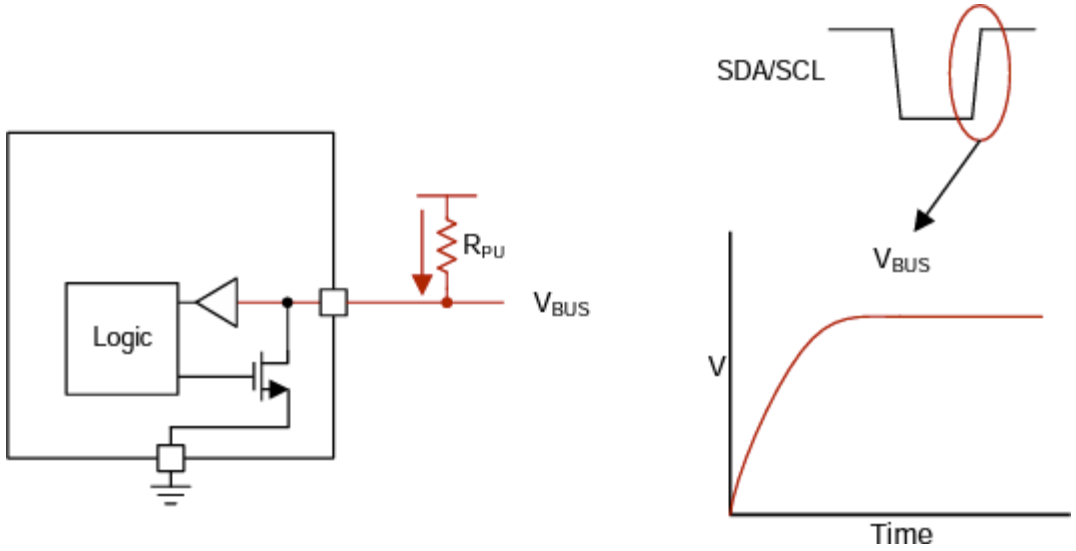
I2C Bus Control – Sending a '0'



Source: Texas Instruments

- If a device needs to send a 0, it pulls down the bus!
- This is fast
- The pull-up resistor limits the current to a safe level for the transistor

I2C Bus Control – Sending a '1'

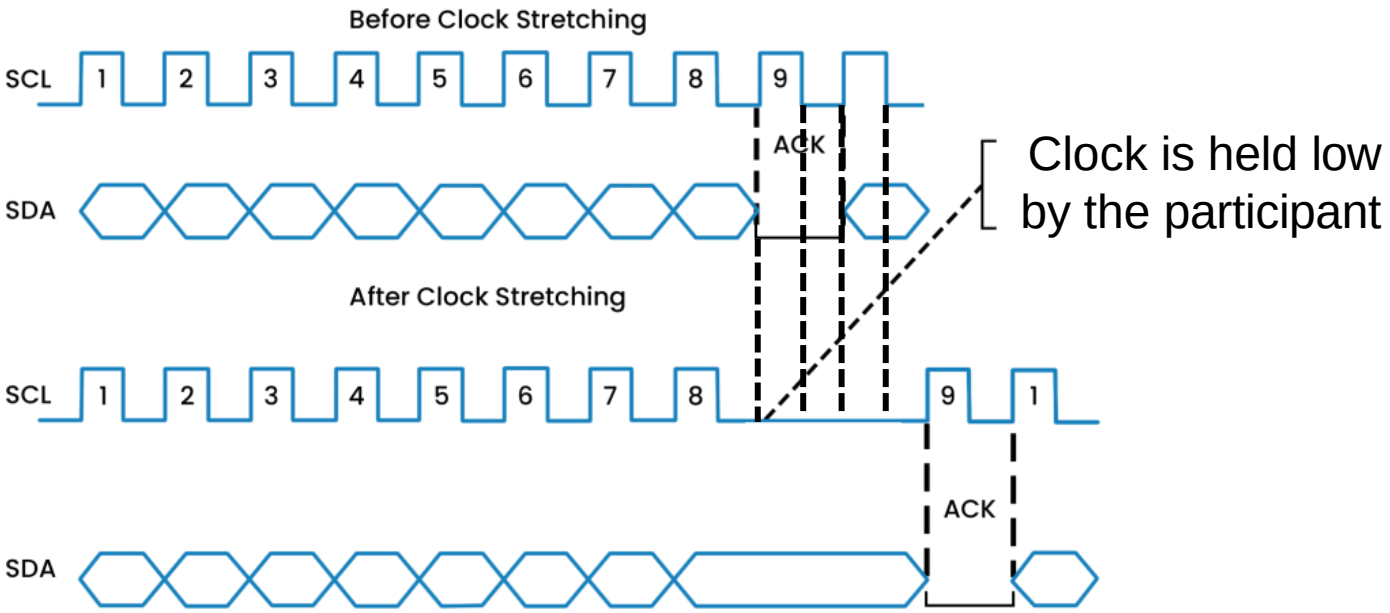


Source: Texas Instruments

- If a device needs to send a 1, it releases the bus!
- The pull-up resistor will charge the bus capacitance…which takes time
- The pull-up resistor needs to be low enough that this happens sufficiently quickly

I²C – Clock Stretching

- The coordinator controls the clock, but the addressed participant may hold the clock line low if it's not ready to process more data yet and needs time to catch up.



- Remember, the clock and data lines are observed on the ground side of the pull-down resistors, so if any participant or coordinator pulls the line down, it will be observed as 0v w.r.t. ground. Only if the resistors aren't grounded will they float high to the full line voltage.

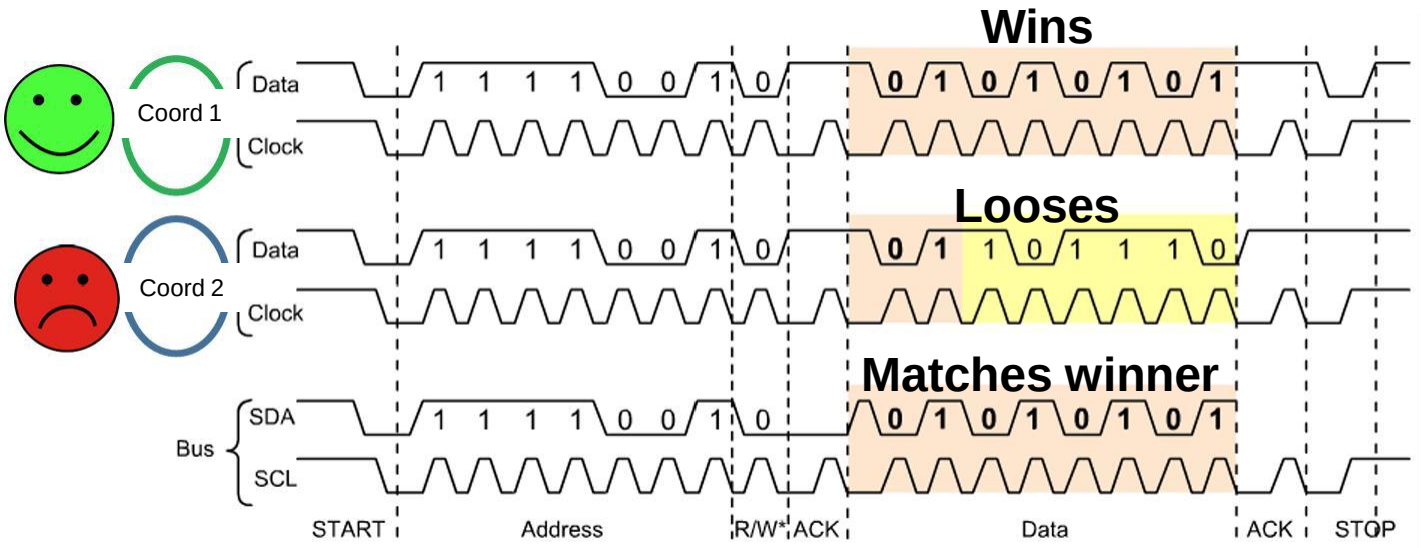
I^2C – Multi-Coordinator Capability

- What if more than one microcontroller/coordinator is connected to the bus and wants to send at the same time?
- Both stations will start to transmit at the same time according to the same clock (We'll talk more about that clock in a moment)
  - Each will pull the data line down and release it according to the bit each wants to send
  - Pulling the line down beats releasing the line as releasing the line means passively allowing it to float high and pulling it down means actively grounding it.
  - Each sending stations are happy so long as the line does what it asks, but if the line is ever pulled down while a station is sending, that station knows another station conflicted with it and that that station won.
    - The station that lost stops transmitting for now and tries again later.
  - This continues until a station finishes its whole transmission
  - Loss-Free collision resolution! No network time was wasted!

I²C – Multi-Coordinator Capability

I2C Bus Arbitration.

- The moment their data bits do not match any more then The Coord 2 loses arbitration and It must backs Off because when Coord 2 tries to move the SDA line high the data on the bus remains low due to wired-AND configuration (as Coord 1 already occupies it.)
.



Very slightly adapted from:

- https://rophoenixmakerevolution.files.wordpress.com/2015%2F09%2Fspi-and-can-bus.pdf

I^2C – Multi-Coordinator Capability

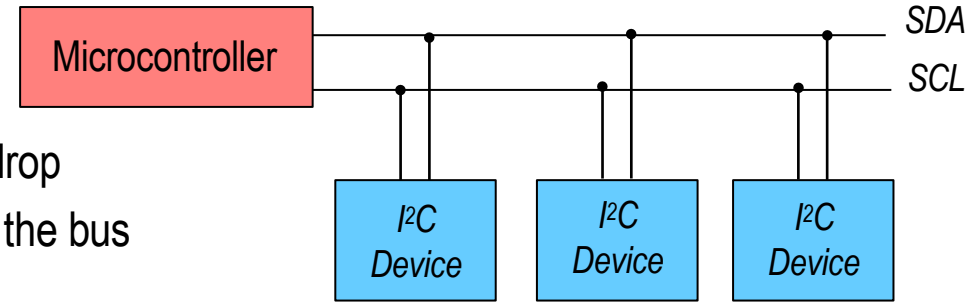
- How do all of the stations see the same clock?
  - The principle is the same as for clock stretching. The clock is only high if all stations allow it to be high, otherwise it is low, because some station has grounded it.
  - So, the clock might not be a perfect square wave. It might spend more time low than high. But, that's okay. The stations react to the rising edges, none-the-less
  - The rising edges won't get more frequent (than the fastest one), since the low phases will just grow larger, not grow more frequent, as they overlap.
  - All stations will see the same rising edges.

I²C – Multi-Coordinator Capability

- Why don't all protocols do this?
  - This works because all stations can see the same rising and falling edges.
  - But it takes time for clock signals to propagate along the wire and to interact with each other
  - If the clock period is too small w.r.t. the propagation time, the overlap will look very different to different stations.
  - I²C is able to do this because the data rate is relatively slow as compared to the physical length of the bus, which controls how long it takes a signal to propagate.
  - Network protocols, such as Ethernet, can't do this for two reasons:
    - There is a bunch higher clock rate to support a much higher data rate, which means that clock period is smaller
    - Which means that the total length of the bus would need to be limited to a smaller maximum, but this doesn't make sense because, unlike I²C, which only needs to cross short distance within a device or small system, more general computer networks often times need to cross much, much longer distances.
  - Each of the two factors above, both independently and together, make this type of collision resolution approach unfit for general purpose computer networks.
  - Computer networks are forced to use destructive/lossy collision resolution, where this type of conflict results in the loss of all data sent during the conflict, not just some of it.

I²C – Summary

- "2-wire" serial protocol
  - SDA (Data), SCL (Clock)
  - Synchronous, Half-Duplex, Multi-drop
  - Uses addressing (7-bits) to share the bus



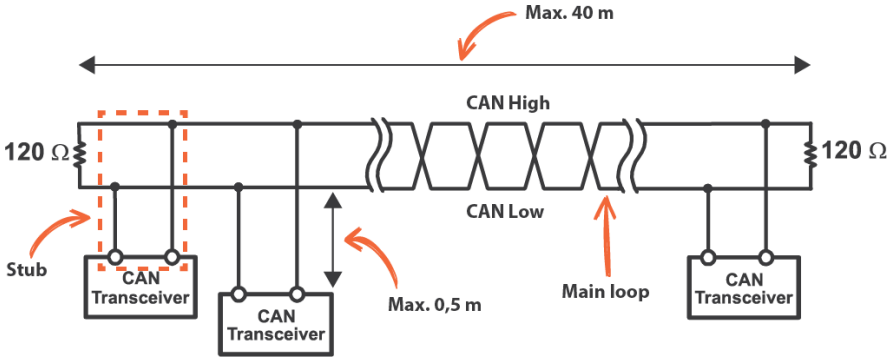
- Advantages
  - Only two wires!
  - Can have multiple devices on a bus (even multiple coordinators are possible)
  - Participants can exercise flow control, i.e. clock stretching
  - Devices can acknowledge messages
- Disadvantages
  - Slower than SPI (similar speeds to UART)
  - Limited to 127 devices on a bus (why?)
  - Needs external hardware (pull-up resistors) to work

Serial Protocol Summary

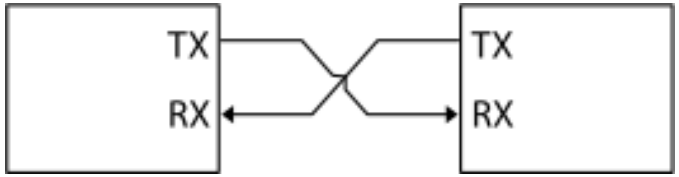
- Serial communication protocols are everywhere!
  - Want to talk to a microcontroller from your computer? UART
  - Want access data from an SD card? SPI
  - Want to finish the lab and pass this course? I²C ☺
- Knowing how to talk about these protocols is useful!
  - What are the advantages/disadvantages of UART?
    - Two wires, full duplex, but only two devices, slower speeds and need to be careful about baud rate mismatch
  - What's the difference between asynchronous/synchronous protocols?
    - Synchronous uses a clock signal!
  - When should I use SPI vs. I²C?
    - SPI is good for high speeds with a low number of devices; I2C is better for many devices at lower speeds

Other Protocols

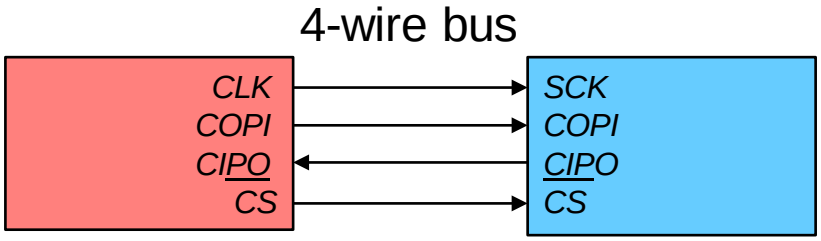
- USB – Universal Serial Bus
  - Succeeded UART as the common peripheral protocol
  - Used for, well, everything
  - Half-duplex (full with USB-3.x), Asynchronous, Differential
- PCI-e – Peripheral Component Interconnect (express)
  - High-speed serial communication bus for use
  - Too complicated to fit on this slide
- CAN – Controller Area Network
  - Common in automotive and robotics applications
  - Half-Duplex, Asynchronous, Addressed, Differential



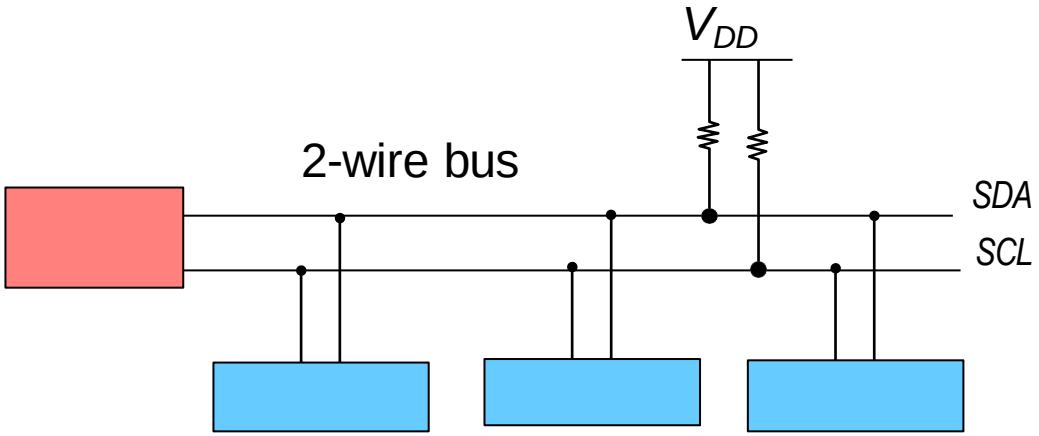
UART


TX
RX
TX
RX

SPI


4-wire bus
CLK
COPI
CIPO
CS
SCK
COPI
CIPO
CS

I2C


V_{DD}
2-wire bus
SDA
SCL

Takeaways

- Many ways to carry out serial communications
  - Synchronous (I2C, SPI) or asynchronous (UART: RS-232/422/485, etc)
  - Full duplex (UART, SPI) or half duplex (I2C)
  - Using a shared bus with device addresses (I2C) or chip select lines (SPI)
- In contrast with wireless communication, which modulates a sinusoidal carrier, these are simply high/low voltages pulled down/up onto wires
- Data rates are limited by mostly limited by **capacitance** on the wires/cables
  - https://www.allaboutcircuits.com/technical-articles/i2c-design-mathematics-capacitance-and-resistance/