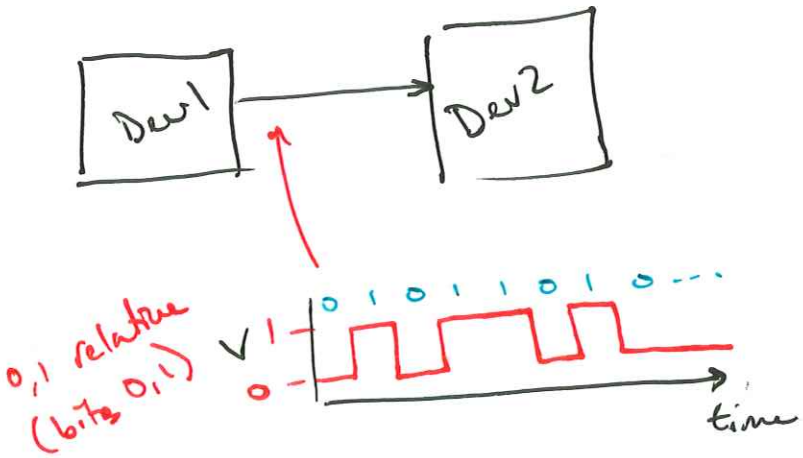
Tommorrow: Quiz 3 (ADC, System Responses, PID)

→ Must be in person!

Communication, specifically: Digital Communication

→ Convey digital data between devices.

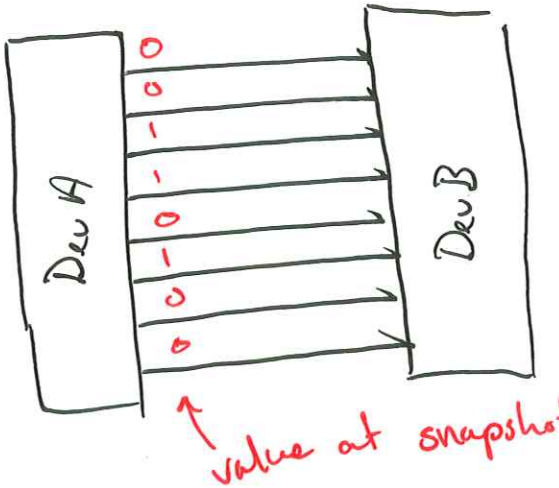


Serial Comms

Bits sent 1 after another

→ sent sequentially.

Simple hardware but limited throughput.



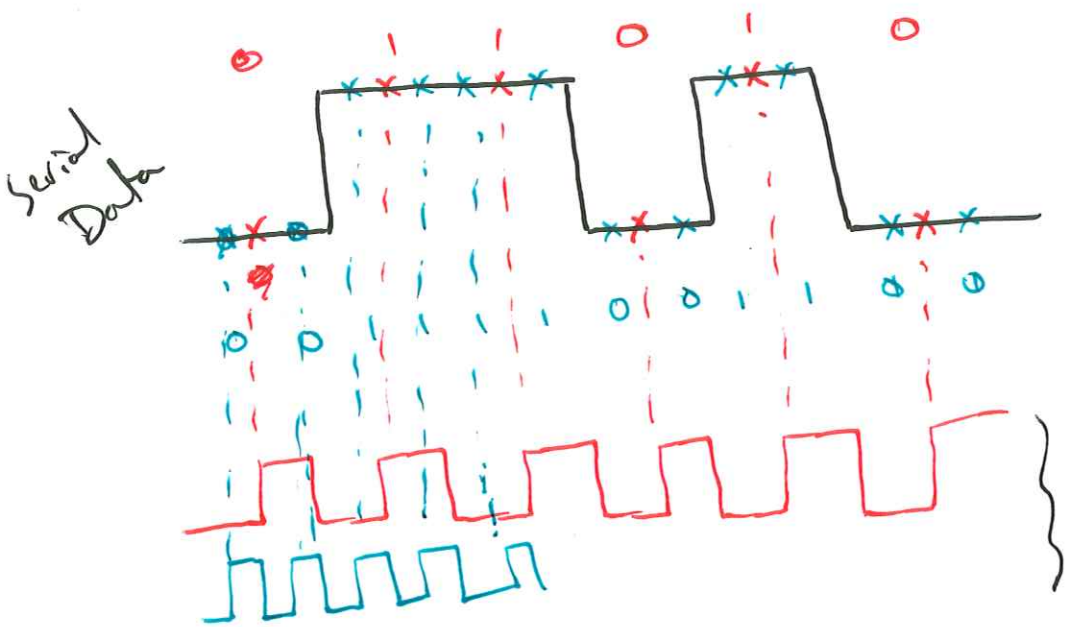
Parallel comms.

Multiple "parallel" lines send multiple bits at once.

for this example: 1 Byte at a time.

More throughput at expense of hardware complexity.

Serial Communication

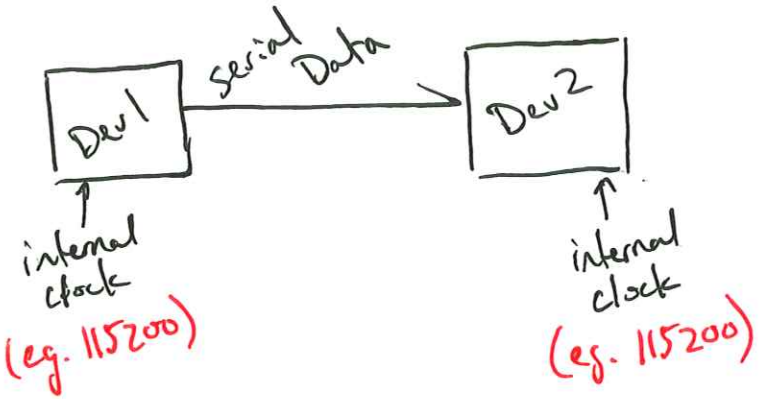


x, x ← sample points

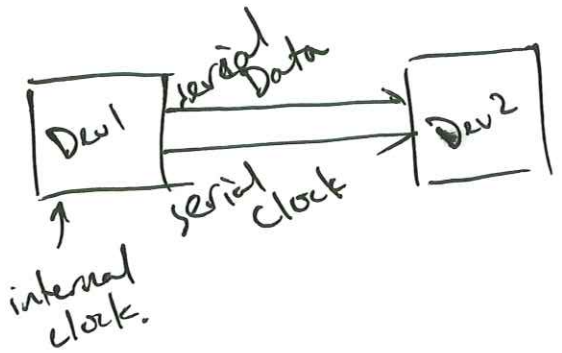
Receiver needs to know this timing!

→ provided by

Asynchronous

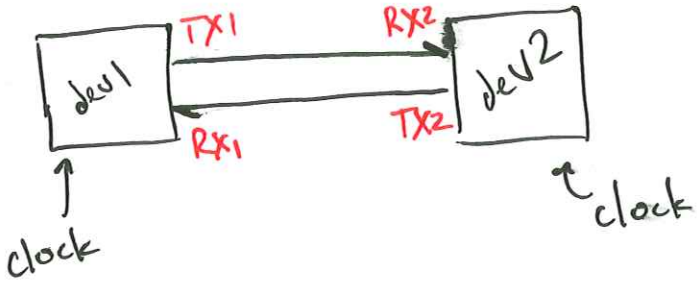


Synchronous



Serial Protocols

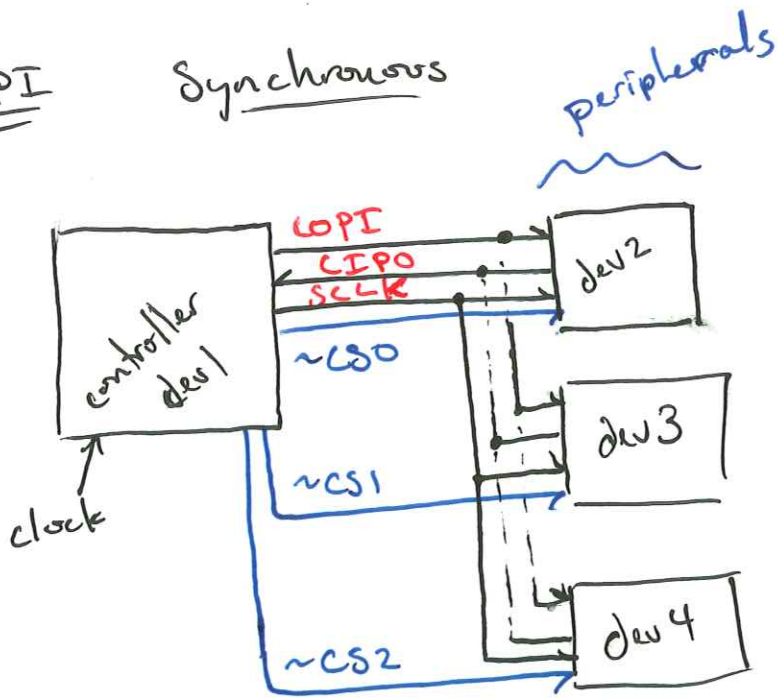
UART (used for printing)
Asynchronous.



full-Duplex:
Data can transmitted in both directions at same time.

Limited to 2 Devices

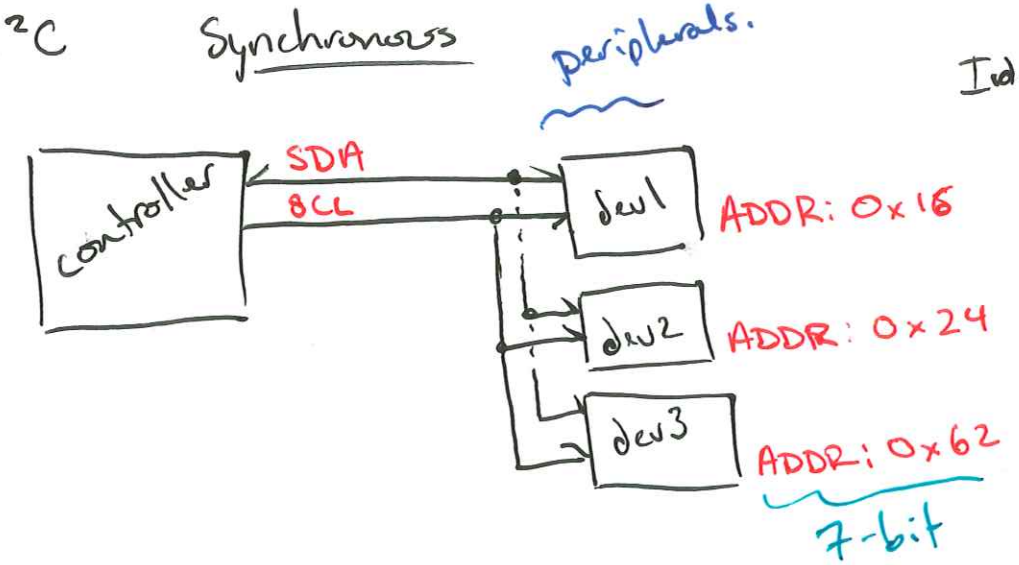
SPI Synchronous



COPI: Controller-Out Peripheral-In
CIPO: Controller-In Peripheral-Out.
SCLK: Serial Clock
~CS#: Chip Select line.
~ means "Active Low"

full Duplex
→ "fast"

I^2C Synchronous



Identification of Active device done through addressing

→ Each device assigned a 7-bit (usually) address.
(can be 10-bit in some cases)
limited to 127 different devices.

Half-Duplex Comms.
Bidirectional ~~but~~ but only 1 direction at a time.

8th bit of 7-bit addresses indicates direction of data flow:
is controller writing data to a peripheral or reading from.

SDA: Serial Data
SCL: Serial Clock

↓

Both are "Open-Drain" outputs.

full addressing:

X X X X X X X X
7-bit addr. R/$\overline{W}$ bit { Read = 1, write = 0

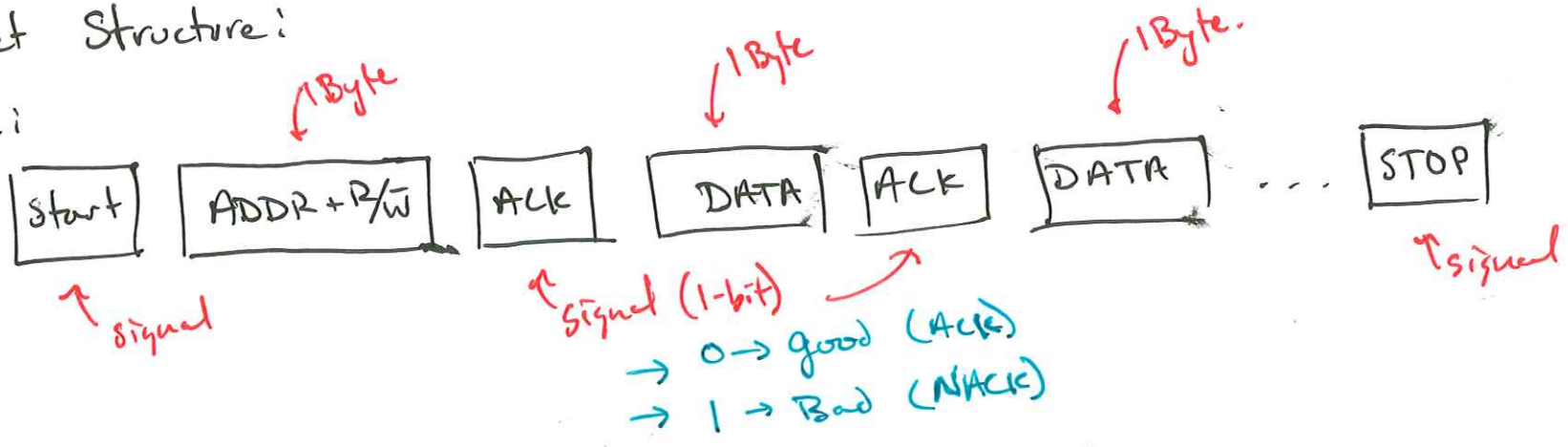
→ Byte is effectively:

(ADDR << 1) + R/$\overline{W}$

→ if ODD: read
EVEN: write

I²C Packet Structure:

Generic:



Data sent is typically identified by "Register" numbering.
for example: Register 0: command, Register 1: sleep, ...

typical write packet:

Start ADDR+R/$\bar{W}$ ACK Start Register ACK Data1 ACK Data2 ACK ... STOP

- ADDR+R/$\bar{W}$: 0 for write
- Start Register: indicates meaning of Data1
- Data1: start register
- Data2: start register +1

RED: Controller
BLUE: Peripheral

Read Packet:

Start ADDR+R/$\bar{W}$ ACK Data1 ACK Data2 ACK ... NACK STOP

- ADDR+R/$\bar{W}$: 1 for read
- Data1: start Reg. #
- Data2: start Reg. +1
- NACK: not required.

for Read: to specify desired data, need to send write packet first with desired start register & no data.

MSPM0 I²C module, Emcon HAL:

Initialization stuff...

uint8_t I2C_writeData(mod, addr, start reg, data, # bytes)

↑ I2C module to use.

↑ must be an array. Of uint8_t

uint8_t I2C_readData (same arguements as above)

→ Read Data is placed within the data array.
starting at index 0!
(write sends from index 0 as well)

eg. Start reg = 4
Data[0] → reg 4
Data[1] → reg 5
⋮

Returned values:

0 - All good
1 - Transmit/write error
2 - Receive/read error

I2C_ReadData does both a write & a read packet.

Issue: Data sent always in bytes
what if value is 2-bytes or 4-bytes...?

Ex. I²C ADC 24-bit. ex output: 0x ABCDEF addr = 0x44

Reg 0: Command
Reg 1: Output Hi Bytes = 0xAB
Reg 2: output Mid Byte = 0xCD
Reg 3: output Low Byte = 0xEf

uint8_t data[3]; ← needs length 3+ for reception of 3bytes.
I2C_readData(I2C0, 0x44, 1, data, 3);

→ Data[0] = 0xAB
Data[1] = 0xCD
Data[2] = 0xEf

uint32_t adc = (data[0] << 16) + (data[1] << 8) + data[2]

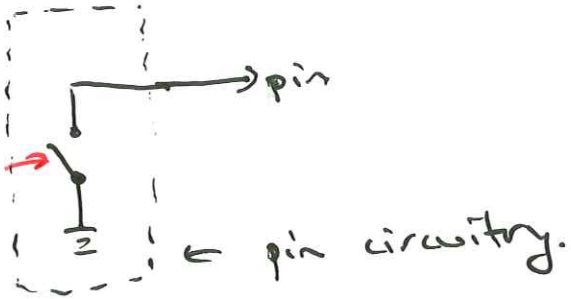
↓ 0x00AB0000

↓ 0x0000CD00

↓ 0x000000Ef

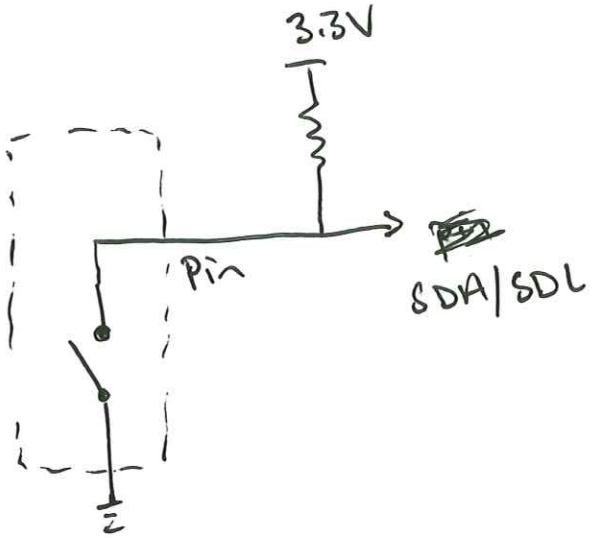
→ adc = 0x00ABCDEF

Open-Drain Outputs:

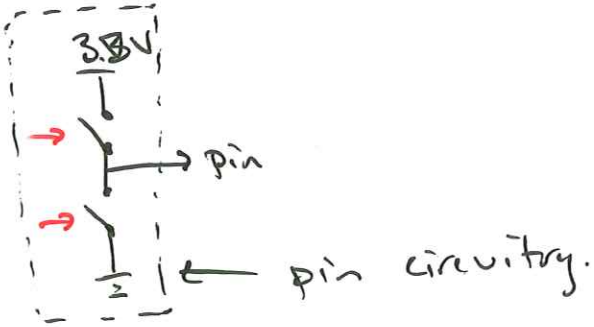


↓

I2C usage:



Push-Pull Outputs: (default)



Benefits:

1) Prevents shorts from 3.3V - GND
2) Isolates different voltages sources (5V does not propagate to other devices, for ex.)
3) Arbitration Purposes.