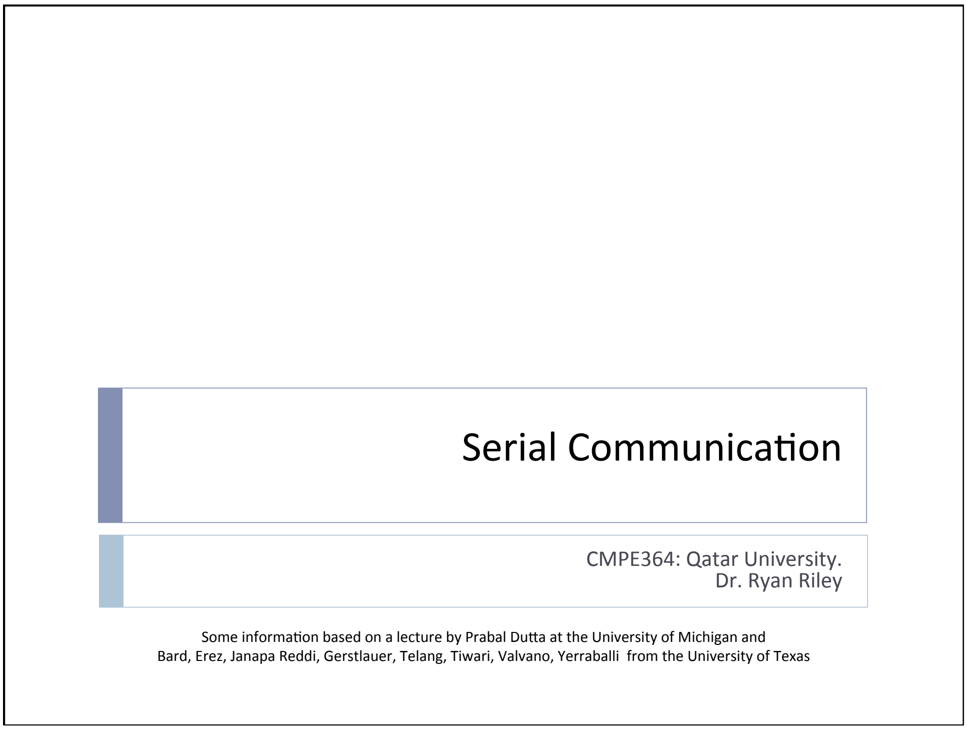
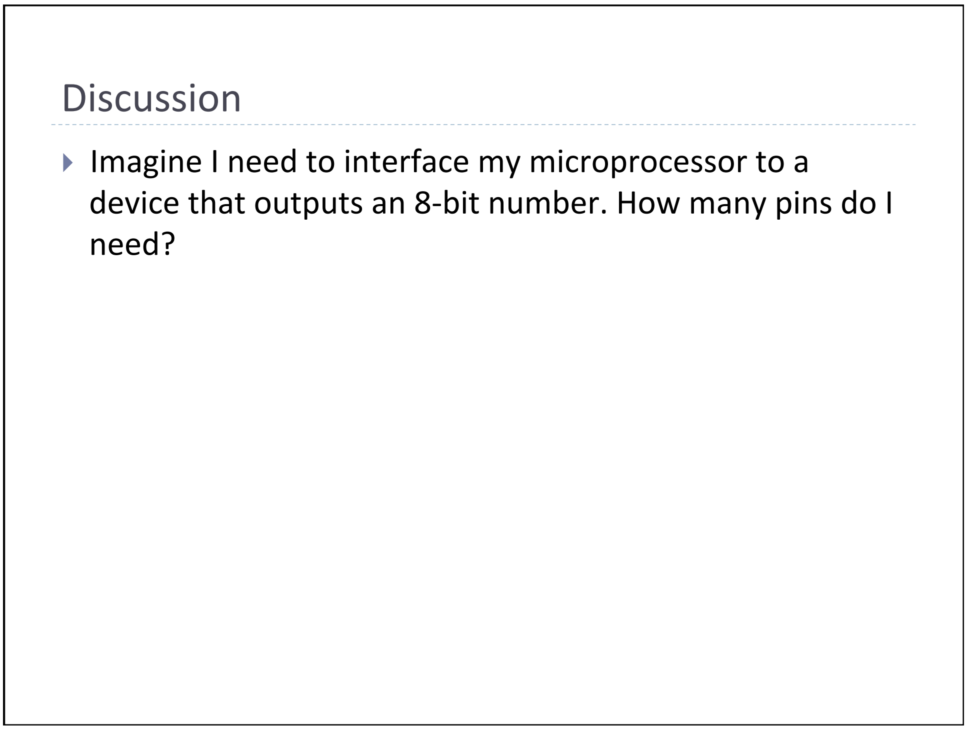


Serial Communication

CMPE364: Qatar University.
Dr. Ryan Riley

Some information based on a lecture by Prabal Dutta at the University of Michigan and
Bard, Erez, Janapa Reddi, Gerstlauer, Telang, Tiwari, Valvano, Yerraballi from the University of Texas



Discussion

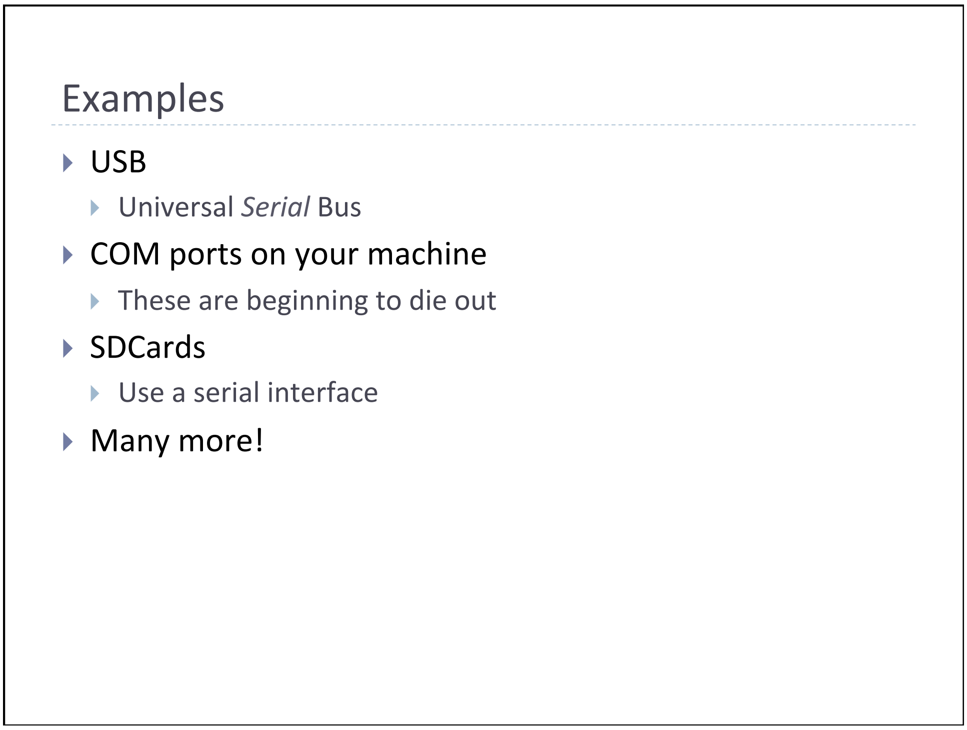
- Imagine I need to interface my microprocessor to a device that outputs an 8-bit number. How many pins do I need?

Intro to Serial Communication

- Serial communication allows us to transmit information using a smaller number of pins/wires.
  - Could be as few as one! (OneWire interface)
  - Frequently 2 or 3
- Imagine a 2 pin system…
  - Pin 1 for Clock
  - Pin 2 for Data
  - One device generates the clock and 1 bit is transmitted every clock cycle.

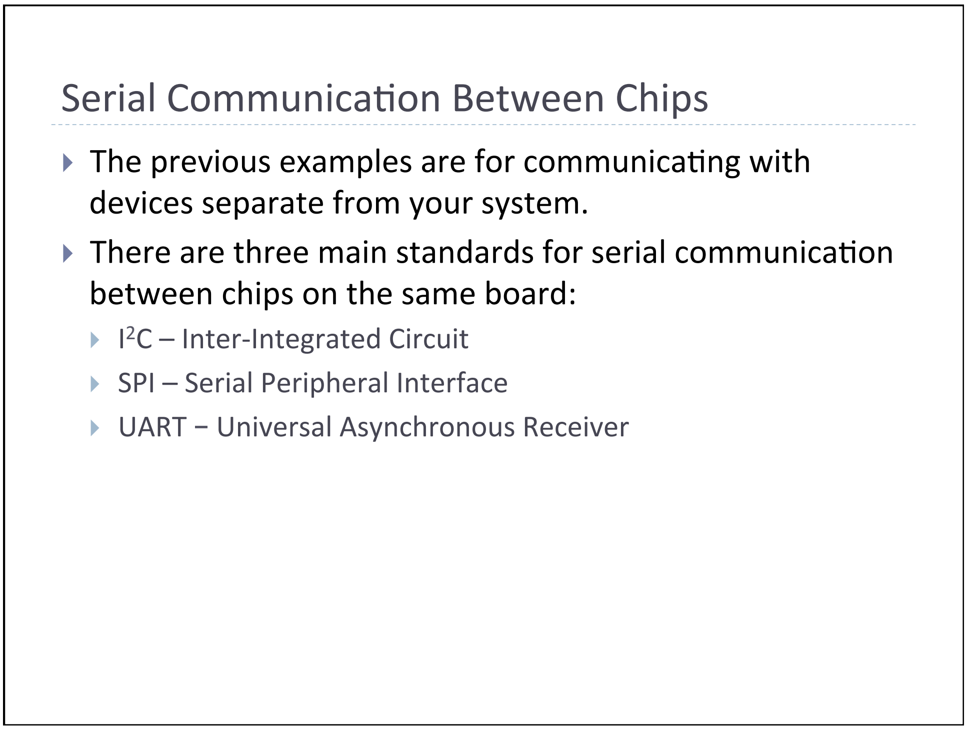
Pros and Cons

- Pros
  - Fewer pins
  - Can handle MANY devices on those pins
- Cons
  - Clock generation/synchronization
  - Slower than transmitting bits in parallel



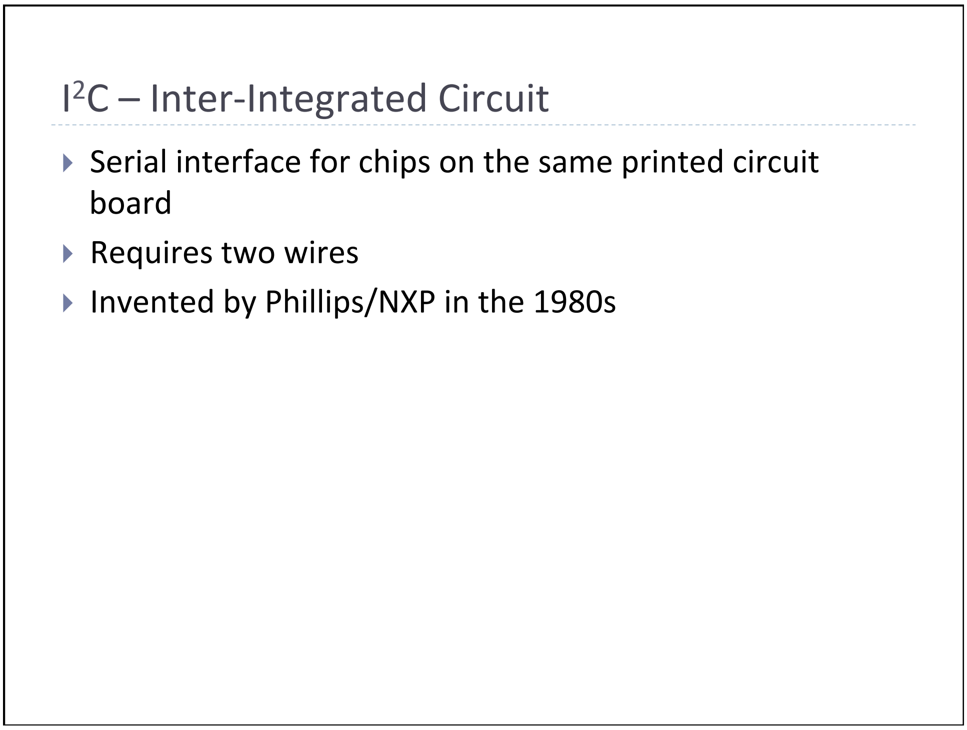
Examples

- USB
  - Universal *Serial* Bus
- COM ports on your machine
  - These are beginning to die out
- SDCards
  - Use a serial interface
- Many more!



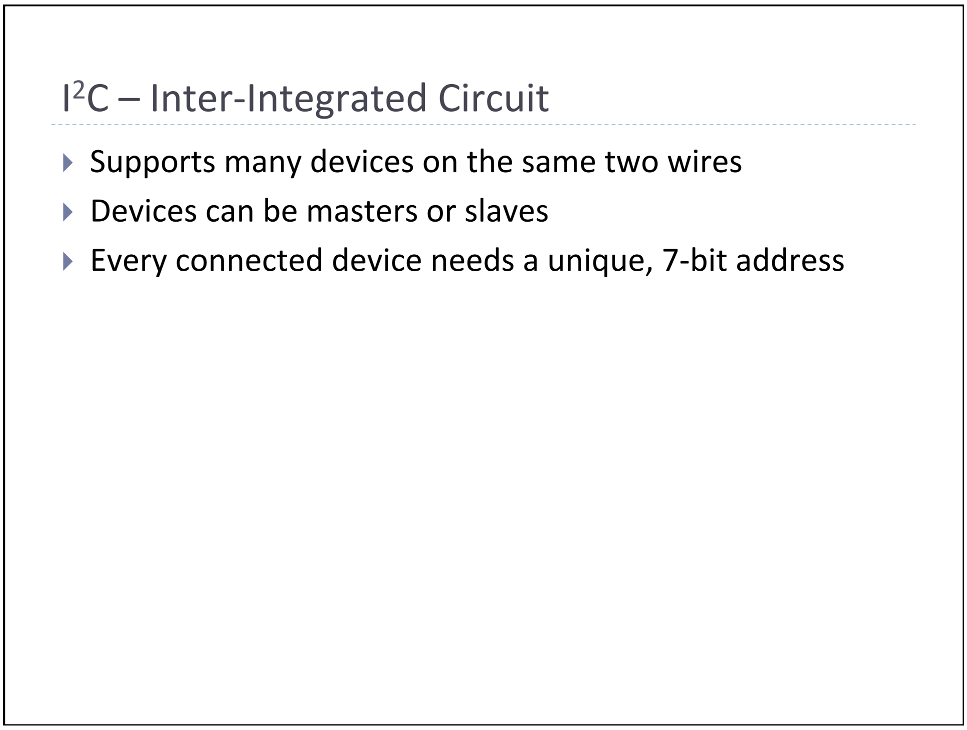
Serial Communication Between Chips

- The previous examples are for communicating with devices separate from your system.
- There are three main standards for serial communication between chips on the same board:
  - I^{2}C – Inter-Integrated Circuit
  - SPI – Serial Peripheral Interface
  - UART − Universal Asynchronous Receiver



I^{2}C – Inter-Integrated Circuit

- Serial interface for chips on the same printed circuit board
- Requires two wires
- Invented by Phillips/NXP in the 1980s



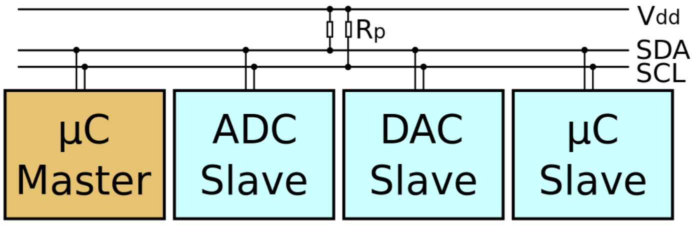
I^{2}C – Inter-Integrated Circuit

- Supports many devices on the same two wires
- Devices can be masters or slaves
- Every connected device needs a unique, 7-bit address

I^{2}C – Inter-Integrated Circuit

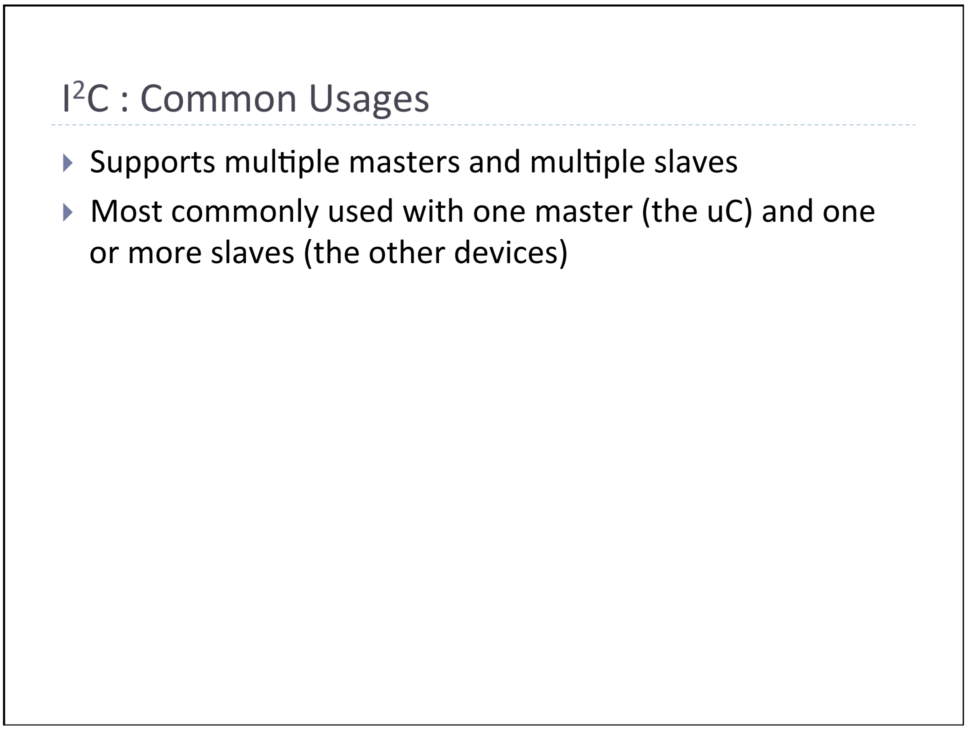
- Maximum speed: 3.4 Mbps
- Minimum speed: None

I^{2}C : Circuit Diagram



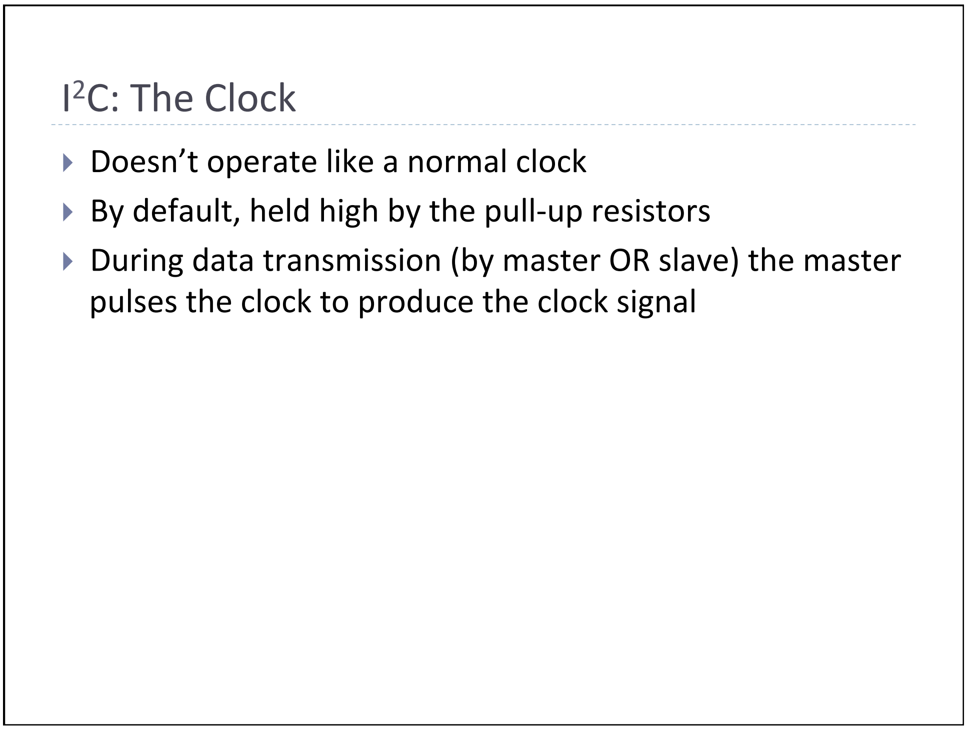
Source: http://en.wikipedia.org/wiki/I2C

- Lines
  - SDA: Serial Data
  - SCL: Serial Clock
- Two pullup resistors
  - Should be between 2K and 10K, depending on the speed you want to transmit



I^{2}C : Common Usages

- Supports multiple masters and multiple slaves
- Most commonly used with one master (the uC) and one or more slaves (the other devices)

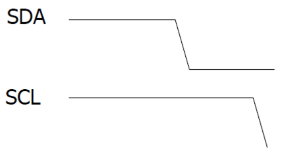


I^{2}C: The Clock

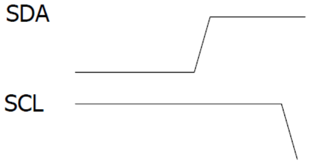
- Doesn’t operate like a normal clock
- By default, held high by the pull-up resistors
- During data transmission (by master OR slave) the master pulses the clock to produce the clock signal

I^{2}C: Starting and Stopping an Operation

- Starting a transmission
  - Master pulls SDA low while SCL is high



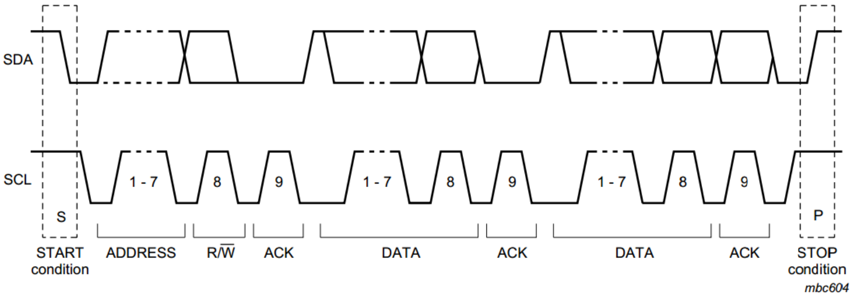
- Stopping a transmission
  - Master pulls SDA high while SCL is high



I^{2}C: Standard Write Operation

1. Master sends start signal
2. Master sends 7-bit device address
3. Master sends 1-bit operation (write)
4. Slave sends ACK
5. Master sends memory address or command
6. Slave sends ACK
7. Master sends 8-bits of data
8. Slave sends ACK
9. Master sends stop signal

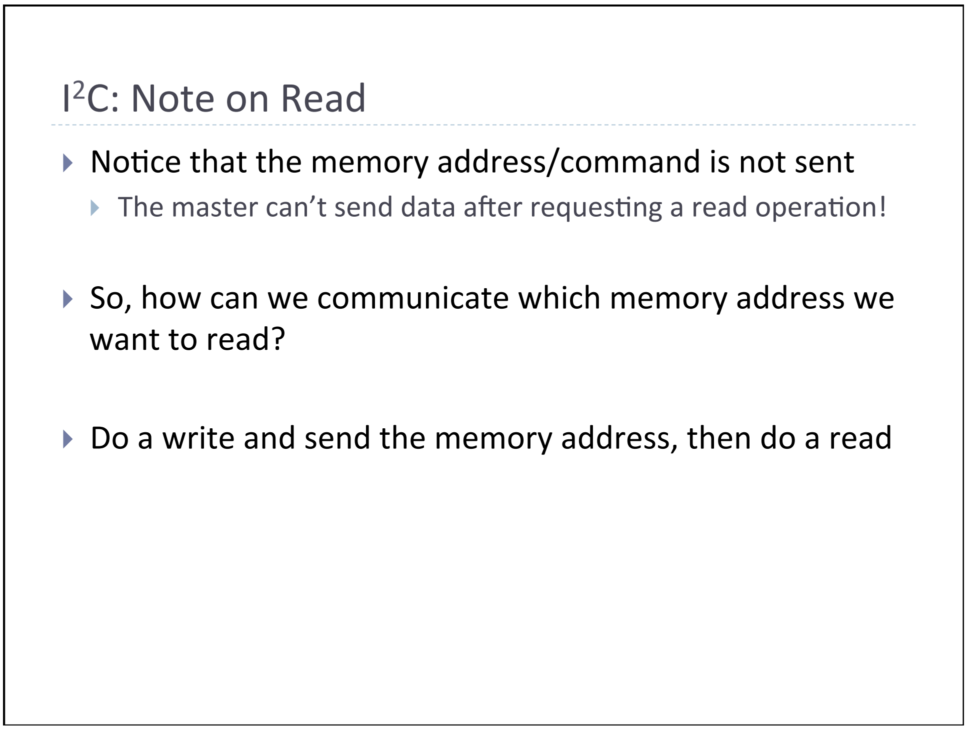
I^{2}C: Timing Diagram



From: http://www.nxp.com/documents/user_manual/UM10204.pdf

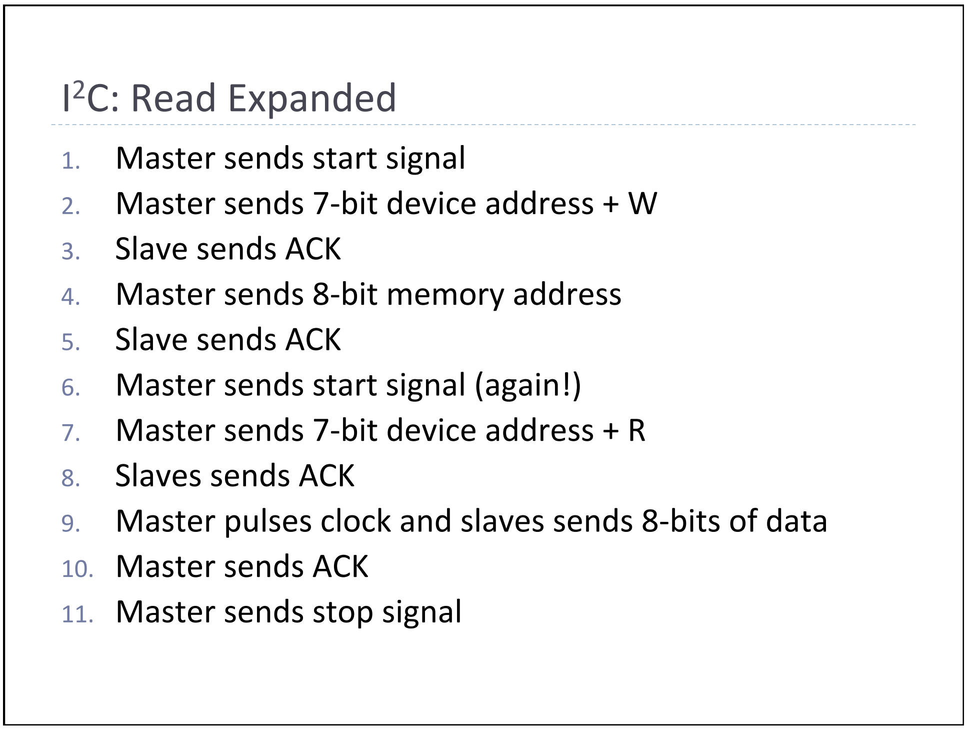
I^{2}C: Standard Read Operation

1. Master sends start signal
2. Master sends 7-bit device address
3. Master sends 1-bit operation (read)
4. Slave sends ACK
5. Master pulses clock and slave sends 8-bits of data
6. Master sends ACK
7. Master sends stop signal



I^{2}C: Note on Read

- Notice that the memory address/command is not sent
  - The master can’t send data after requesting a read operation!
- So, how can we communicate which memory address we want to read?
- Do a write and send the memory address, then do a read

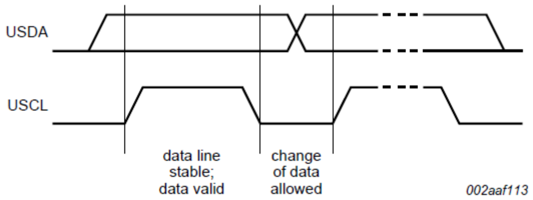


I^{2}C: Read Expanded

1. Master sends start signal
2. Master sends 7-bit device address + W
3. Slave sends ACK
4. Master sends 8-bit memory address
5. Slave sends ACK
6. Master sends start signal (again!)
7. Master sends 7-bit device address + R
8. Slaves sends ACK
9. Master pulses clock and slaves sends 8-bits of data
10. Master sends ACK
11. Master sends stop signal

I^2C: Usage Details

- Other than START and STOP, SDA should only be changed while SCL is low



From: http://www.nxp.com/documents/user_manual/UM10204.pdf

I^2C: Usage Details

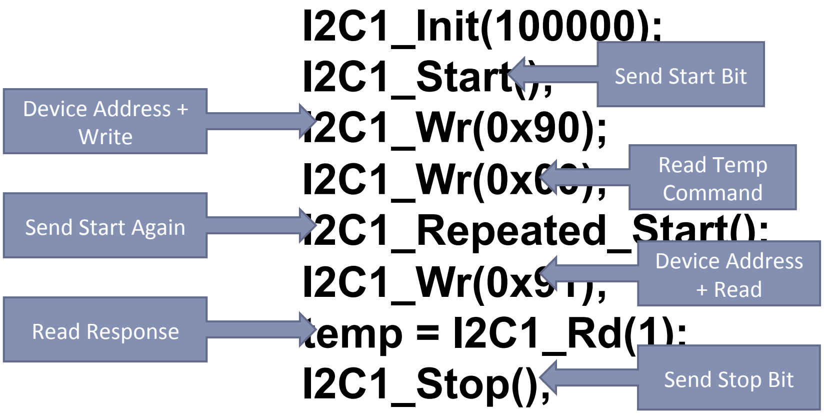
- Data is always sent 8-bits at a time
- You can send multiple 8-bit values one after another

I^{2}C: Usage Details

- Where do I get the device address?
  - From the device's data sheet
  - Some devices make it configurable via pins
- How do I know the proper memory addresses/command to send?
  - From the device's data sheet
- How do I know what clock rate to use?
  - From the device's data sheet. (Or, when in doubt, use a slow one like 100 KHz)
- Lesson: Read your device's data sheet!

I^{2}C: Sample Psuedo Code

- Device address is… 1001 000



```
I2C1_Init(100000);
I2C1_Start();
I2C1_Wr(0x90);
I2C1_Wr(0x00);
I2C1_Repeated_Start();
I2C1_Wr(0x91);
temp = I2C1_Rd(1);
I2C1_Stop();
```

Send Start Bit

Device Address + Write

Read Temp Command

Send Start Again

Device Address + Read

Read Response

Send Stop Bit

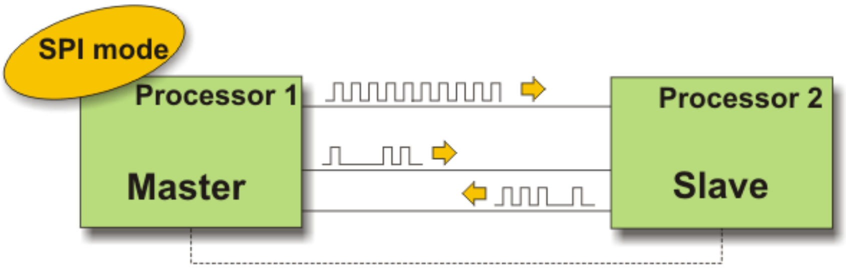
I^{2}C: mbed

- https://developer.mbed.org/handbook/I2C

SPI: Serial Peripheral Interface

- From Motorola
- Operates in full-duplex mode
  - Data can be sent both directions at the same time
- Uses a master/slave model
- Supports multiple slaves

SPI: Simplified Diagram

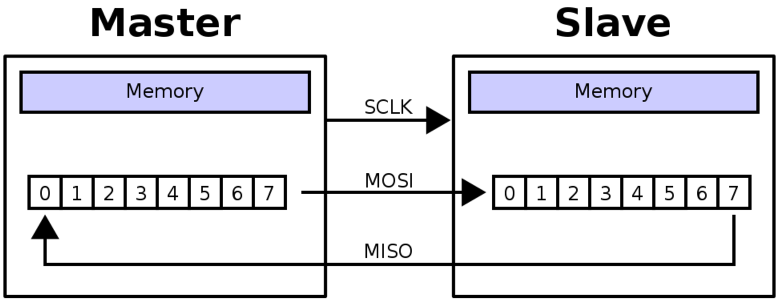


SPI: Basic Operation

- Master controls clock
- Each clock cycle…
  - Master sends 1-bit to slave
  - Slave sends 1-bit to master
- This is full-duplex communication

SPI: Full-Duplex

- Data can be sent and received at the same time
- Typically done with a shift register



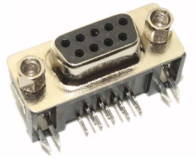
Source: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

SPI: mbed

- https://developer.mbed.org/handbook/SPI

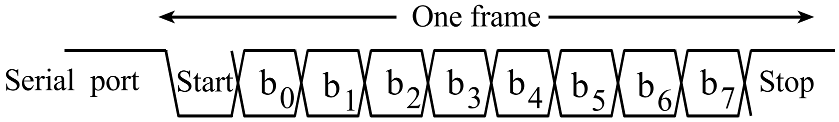
UART - Universal Asynchronous Receiver

- One of the initial serial communication standards in computers



- In the previous two interfaces there was a common clock
  - UART has no common clock

UART Frame



- By default, the signal is high
  - Design choice taken from the telegraph
- Pulled low to “start”
- Send 8 bits
- Set high to “stop”

UART – Setting the speed

- With no common clock, how do you decide how fast to send the bits?
- Speeds are standardized
- Both sides need to be programmed to assume the same speed
- Common rates:
  - 1200 bps
  - 2400 bps
  - 4800 bps
  - …
  - 115,200 bps
  - …
  - 460,800 bps

UART – Parity

- There is the option to send a extra bit for parity
  - Check for errors
- The parity bit is set based on the other bits
  - P = 1 if the number of 1s sent is odd
  - P = 0 if the number of 1s sent is even
- This lets you detect simple errors

UART - Settings

- You need to preconfigure both sides with speed, bits, parity, etc.
- Format:

  Speed (bps), Parity, # of Bits, # Stop Bits
- Example:

  9600, N, 8, 1

UART - mbed

- https://developer.mbed.org/handbook/Serial