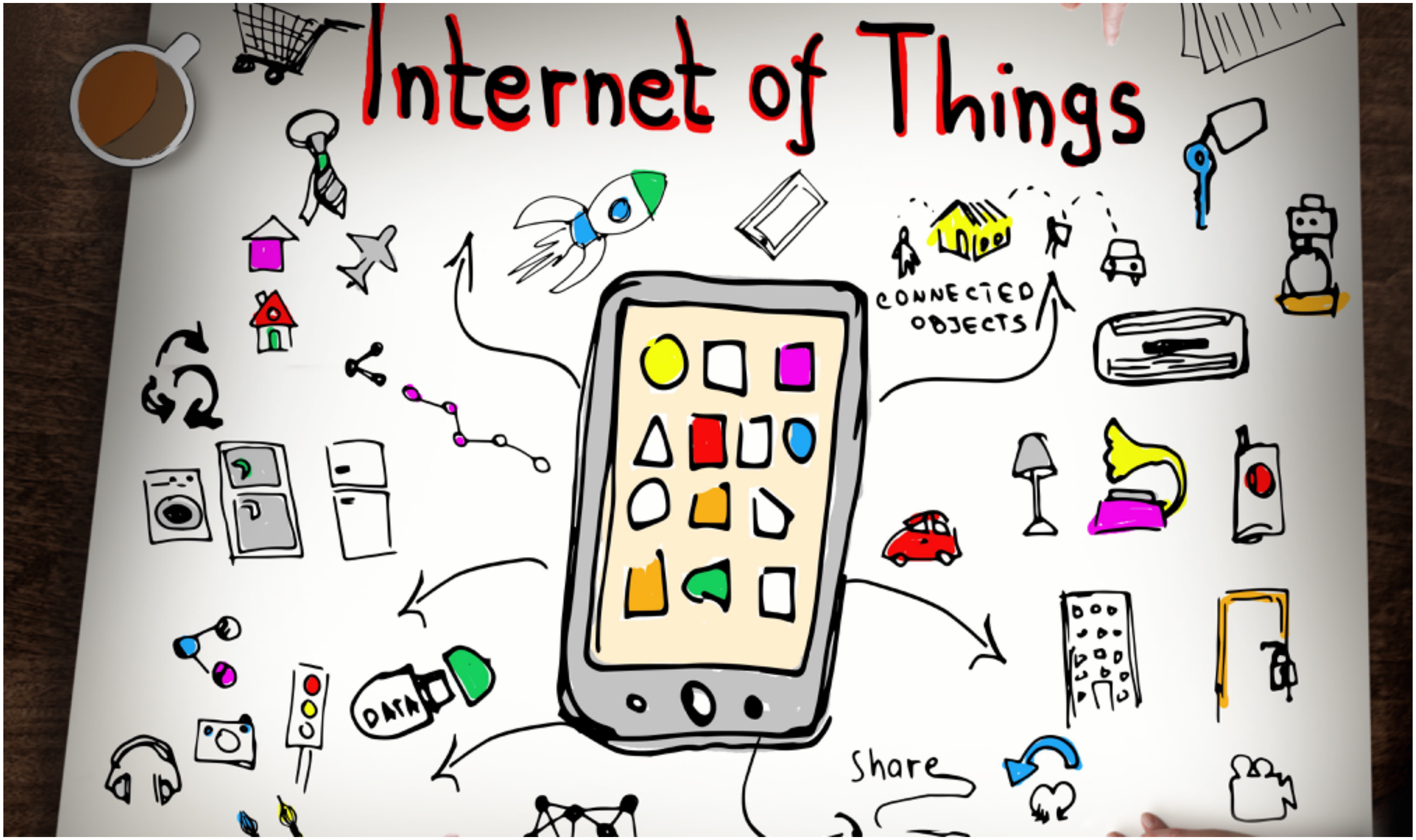


IoT Protocols: I2C, SPI

18-738 Sports Technology

03/17/2022

Priya Narasimhan
ECE Department
Carnegie Mellon University
@yinzcampriya

Outline of This Lecture

- I2C bus
- Master-slave protocol
  - Sending data
  - Acknowledgment
- Multi-master protocol
  - Arbitration
- Serial Peripheral Interconnect (SPI)

Background

- I2C is also written as I^{2}C (pronounced “eye-squared-see” or “eye-two-see”)
  - Stands for Inter-Integrated Circuit (IIC)
- Two-wire party-line bus for “inside the box” communication

party-line bus = broadcast

inside the box because IIC started in Old TV Sets, the TV sets usually have giant motherboard.

- Intended for short-range communication between ICs on a circuit board or across boards in an embedded system

Old TV sets motherboard includes remote, rendering chip, clock,etc.

- I2C devices commonly used in industrial applications
  - EEPROMs, thermal sensors, real-time clocks, RF tuners, video decoders/encoders
- Philips Semiconductors is the primary champion of I2C
  - Specification publicly available at http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf
  - Originally developed for communication between devices inside a TV set in the mid-1980s

Purpose

- Designed by Philips ~20 years ago
- Original purpose was to allow easy communication between components which resided on the same circuit board

I2C is a single line bus. Allows you to plug and play multiple devices. A combination of hardware and software protocal.

- Combines hardware and software protocols to provide a bus interface that can connect many peripheral devices
- I2C is now not only used on single boards, but also to connect components which are linked via cable
- All I2C-compatible devices have an on-chip interface that allows them to communicate directly with each other via the I2C-bus
- Supports easy, ready-to-use interfacing of various boards and digital circuits (even if they are independently designed)
- Allows for "plug-and-play" and evolution of ICs into a larger system

I2C used to use for ICs on the same board. Now I2C is used for multiple boards as well.

Characteristics of I2C

- Only two bus lines are required
- Each device connected to the bus is software-addressable by a unique address and
  - Simple master/slave relationships
- True multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer

In Bluetooth Piconet, you can only have one master.
In I2C, multiple masters are allowed but there's master arbitration, only one can be master at a time.

- Serial, 8-bit oriented, bidirectional data transfers
  - Slow: Up to 100 kbit/s in the standard mode
  - Fast: Up to 400 kbit/s in the fast mode
  - High-speed (3.4 Mbps), I2C version2.0
- On-chip filtering rejects spikes on the bus data line to preserve data integrity

If you see giant spike, you don't see 1 flips to 0 or 0 flips to 1.

- Limited to about 10 feet for moderate speeds

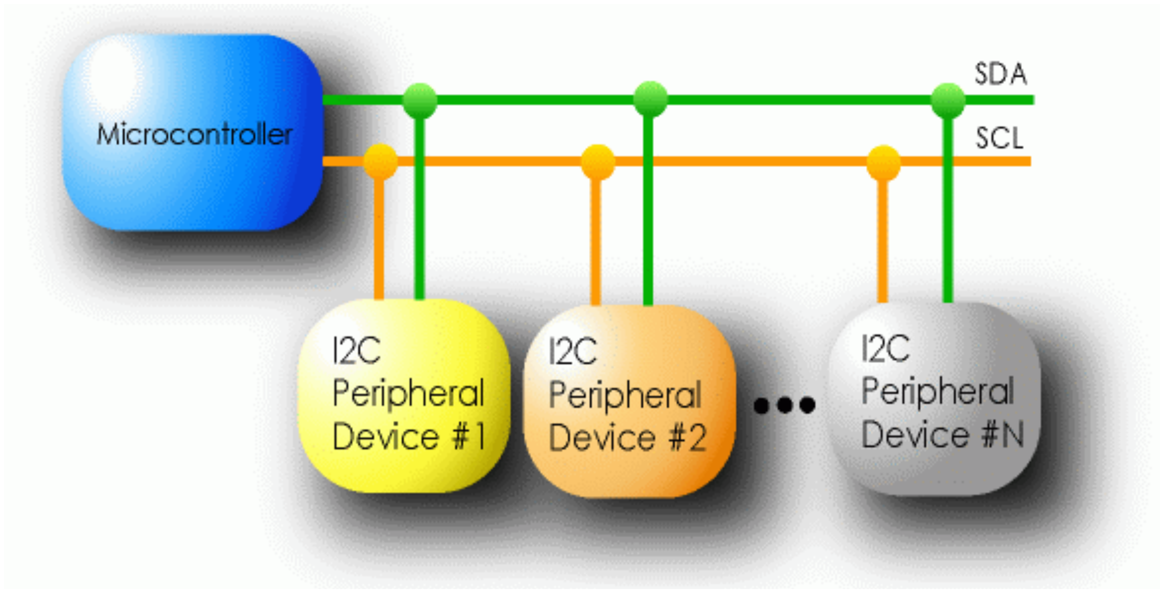
Design Criteria for I2C

- First of all, this is a serial bus
  - Targeting 8-bit microcontroller applications
  - Serial vs. parallel – anyone remember pros and cons?

- Criteria for design of I2C
  - Need to avoid confusion between connected devices Need addressing system, unique device. Addressing.
  - Fast devices must be able to communicate with slow ones Devices with different clock speeds. Flow Control.
  - Protocol must not be dependent on the devices that it connects Device from one company can talk to device from another company.
  - Need to have a mechanism to decide who controls the bus and when Bus arbitration.
  - If different devices with different clock speeds are connected, the bus clock speed must be defined Make sure the overall bus speed speed is defined.

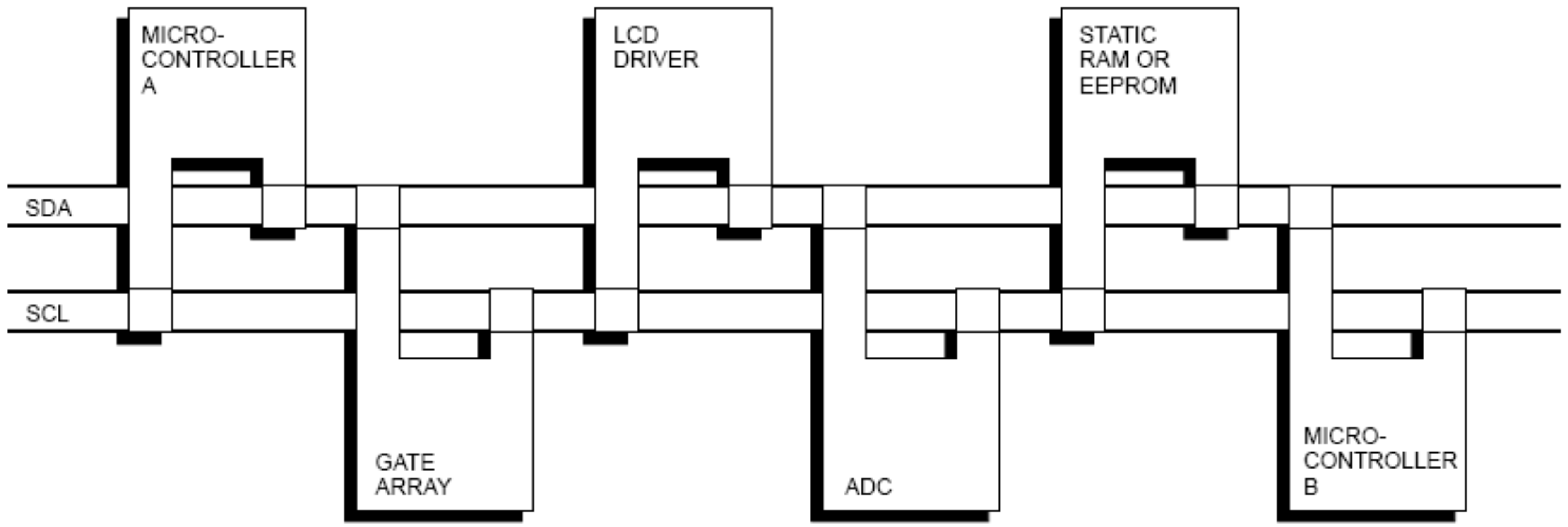
Details

- Two lines: Serial data line (SDA) & serial clock line (SCL)
- Each I2C device recognized by a unique address
- Each I2C device can be either a transmitter or receiver Bi-Direaction communication is possible.
- I2C devices can be masters or slaves for a data transfer
  - Master (usually a microcontroller): Initiates a data transfer on the bus, generates the clock signals to permit that transfer, and terminates the transfer
  - Slave: Any device addressed by the master at that time
  - Roles/relationships are not permanent Master / Slave is not fixed relationship.



Example I2C-Connected System

Devices are just "clip on" to the circuit board. You clip-on and clip-off and it's ready to play.



The notion of "bus", in Bluetooth there's no notion of something controlling a clock line, no clock line in bluetooth.
In Bluetooth, it take turns and frequency hopping instead of a clock line.

Infrared, there's no clock line. Communication is point to point.

Example I2C-connected system with two microcontrollers

(Source: I2C Specification, Philips)

Master-Slave Relationships

- Masters can operate as master-transmitters or master-receivers

  Microcontroller can talk to each other.

- Suppose microcontroller A wants to send information to microcontroller B
  - A (master) addresses B (slave)

    Master is the merely that initiates the data transfer.
    The master-slave is orthogonal to transmitter-receiver.
  - A (master-transmitter), sends data to B (slave-receiver)
  - A terminates the transfer.

    A master doesn't has to be a transmitter.
    It can be a master-receiver with a slave transmitter.
- If microcontroller A wants to receive information from microcontroller B
  - A (master) addresses microcontroller B (slave)
  - A (master-receiver) receives data from B (slave-transmitter)
  - A terminates the transfer

- In both cases, the master (microcontroller A) generates the timing and terminates the transfer

Master always generate the timing.

Multi-Master Capability

Basically Bus Arbitration.

Multi-Master means cases if two microcontroller needs to communicate with the same slave, etc.

- Clearly, more than one microcontroller can be connected to the bus
  - What if both microcontrollers want to control the bus at the same time?
- Multi-master I2C capability supports this without corrupting the message
- Wired-AND connection of all I2C interfaces to the bus for arbitration
- If two or more masters try to put information onto the bus, the first to produce a 'one' when the other produces a 'zero' loses
- Clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line

Bus Arbitration can take any form but in this case it takes the 'first one producing one' form.

- Generation of clock signals on the bus
  - Each master generates its own clock signals when transferring data on the bus
  - A master's bus clock signals can be altered when stretched by a slow-slave device holding down the clock line, or by another master during arbitration

I2C bus arbitration can take different form, in this case it just take this form. A wired-and circuitry to the CLOCK line.
Each master generate its own clock to communication.
"Clock stretching": each master providing their own clock so the clock bus has been stretched.

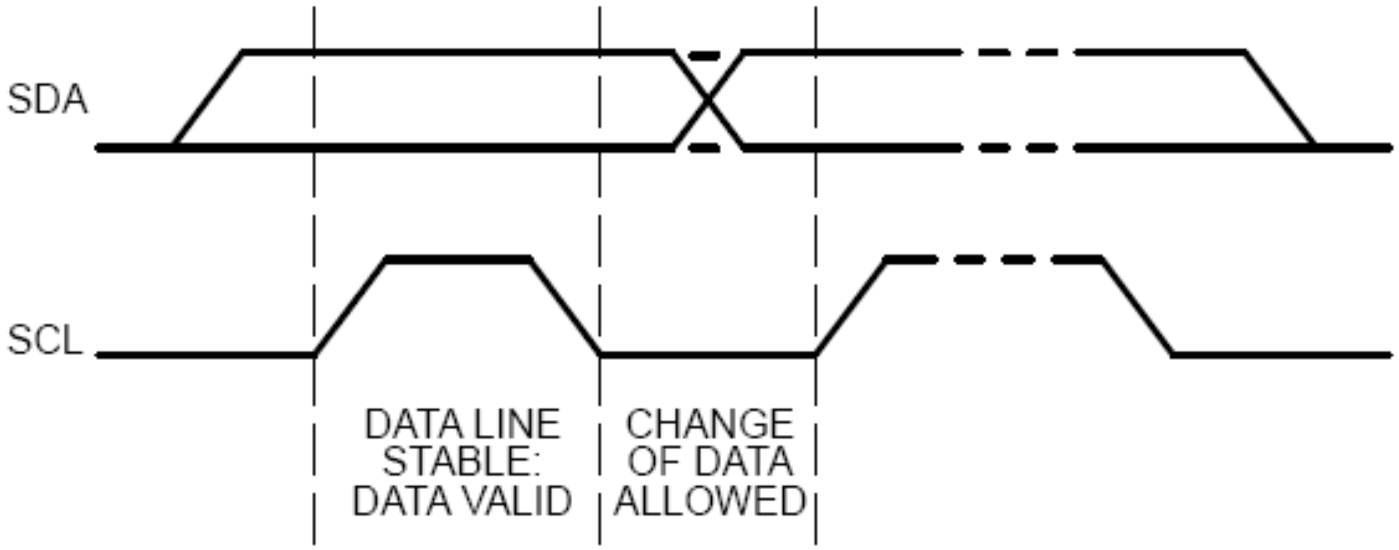
Transferring a Single Bit

- Absolute levels of logical ‘0’ (LOW) and ‘1’ (HIGH) depend on the associated level of VDD

The clock line tells you when to tell to grap a bit from data line.
When clock is low, the data can change; when clock is high, data line cannot change thing. One clock pulse for each bit of data. Everything is logic driven. This is to make sure data integrity.

- One clock pulse is generated for each data bit transferred
- Data on SDA line must be stable during *the HIGH period of the clock*
- HIGH/LOW state of the data line can only change when the clock signal on the SCL line is LOW

One clock pulse for every bit of data.
The data can only change when SCL goes low.
When SCL is high, data is read, fetched
SCL is low, we change data.



START and STOP Conditions

Start and Stop condition always generated by the master device.

- START condition

  RS232 we also have start and stop.

  - HIGH to LOW transition on the SDA line *while SCL is HIGH*

  Usually when SCL is high, SDA is not supposed to change, but if it changes, then it's a start or stop condition.

- STOP condition
  - LOW to HIGH transition on the SDA line *while SCL is HIGH*
- START and STOP conditions are always generated by the master
- Bus considered to be busy after the START condition

  Wait for the data to settle down.

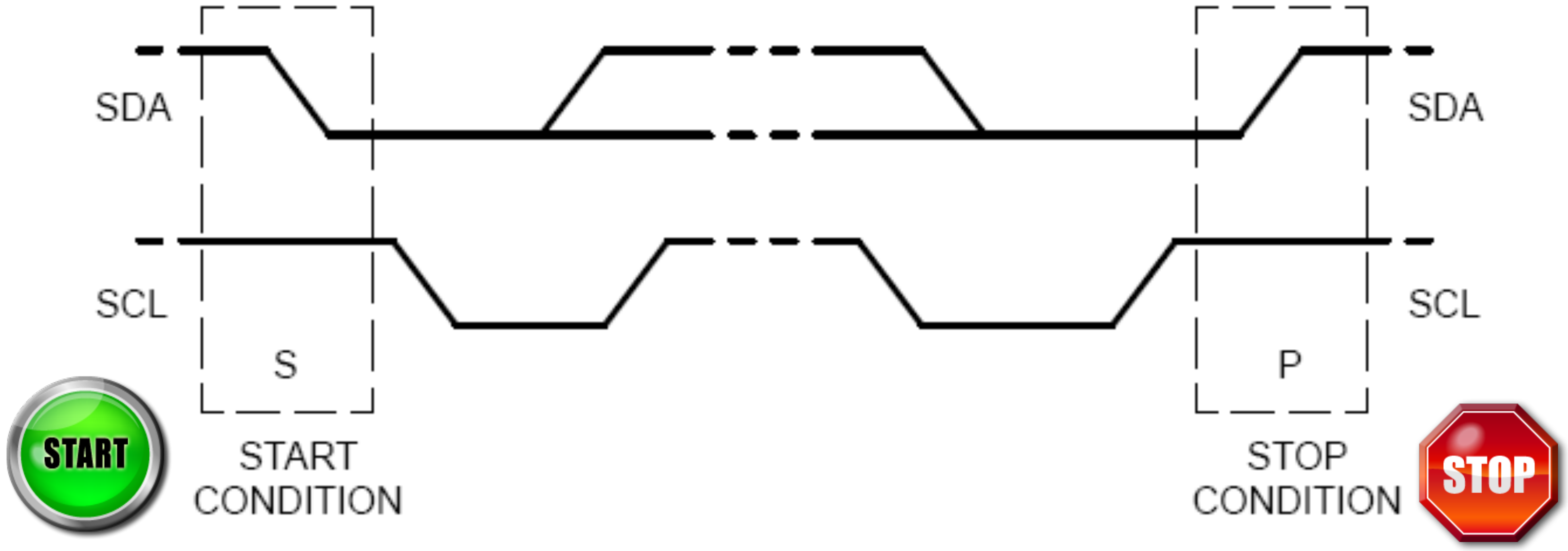
  - Considered to be free again a certain time after the STOP condition

When the SCL is high, and ther's change in SDA, then that's either start or stop.

Start condition: SDA high to low when SCL high

Stop condition: SDA low to high when SCL high.

Legitimate transition: NO SDA transition on SCL high.



The receiver can hold the line low when if the transmitter is too fast and it cannot take the data.
The receiver can force the transmitter to stop and hold the CLK low.
Receiver can decide the the bus speed.

Transferring Data

- Every byte placed on the SDA line must be 8-bits long
  - Each byte has to be followed by an acknowledge (ack) bit
  - Data is transferred with the most significant bit (MSB) first
- Number of bytes that can be transmitted per transfer is unrestricted
- If a receiver can't receive another complete byte of data temporarily (e.g., busy servicing an internal interrupt)
  - Receiver can hold SCL LOW to force the transmitter into a wait state
  - Data transfer resumes when receiver is ready for another byte and releases SCL

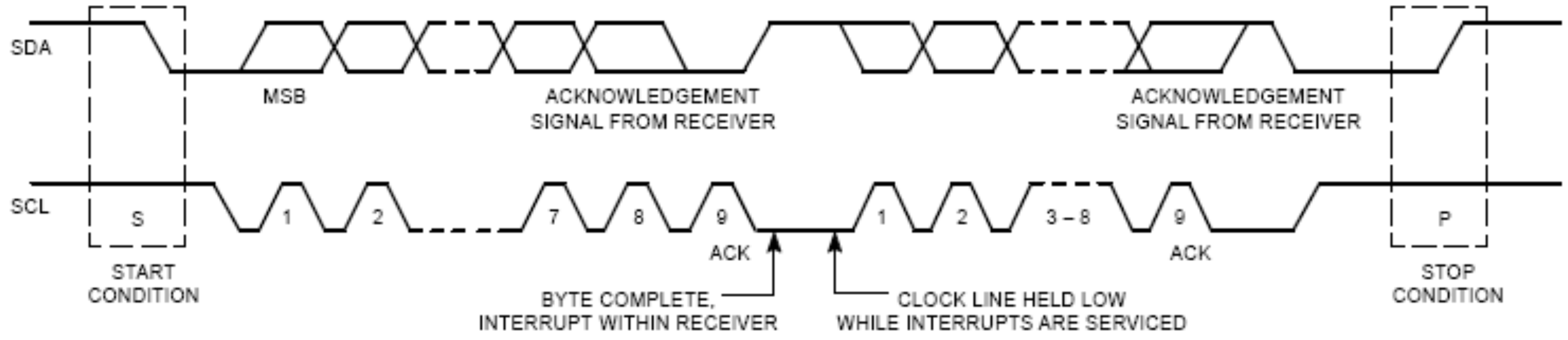
8e2 = 8 bits, even parity, 2 bit check bit

8 bits + 1 bit ack

Sending 9 bits for each byte data sent.

If receiver is really slow, then it is atrocious. If SCL does not change, then no data is transferred: this is clock stretching. This is slow because the receiver keep holding the clock line low.

You can have any numbers of bytes send between start and stop (8 legit bits + 1 ACK). You are sending 9 bits for each bytes of data.
START -- 9 BIT - 9 BIT - 9 BIT...... - 9 BIT -- STOP



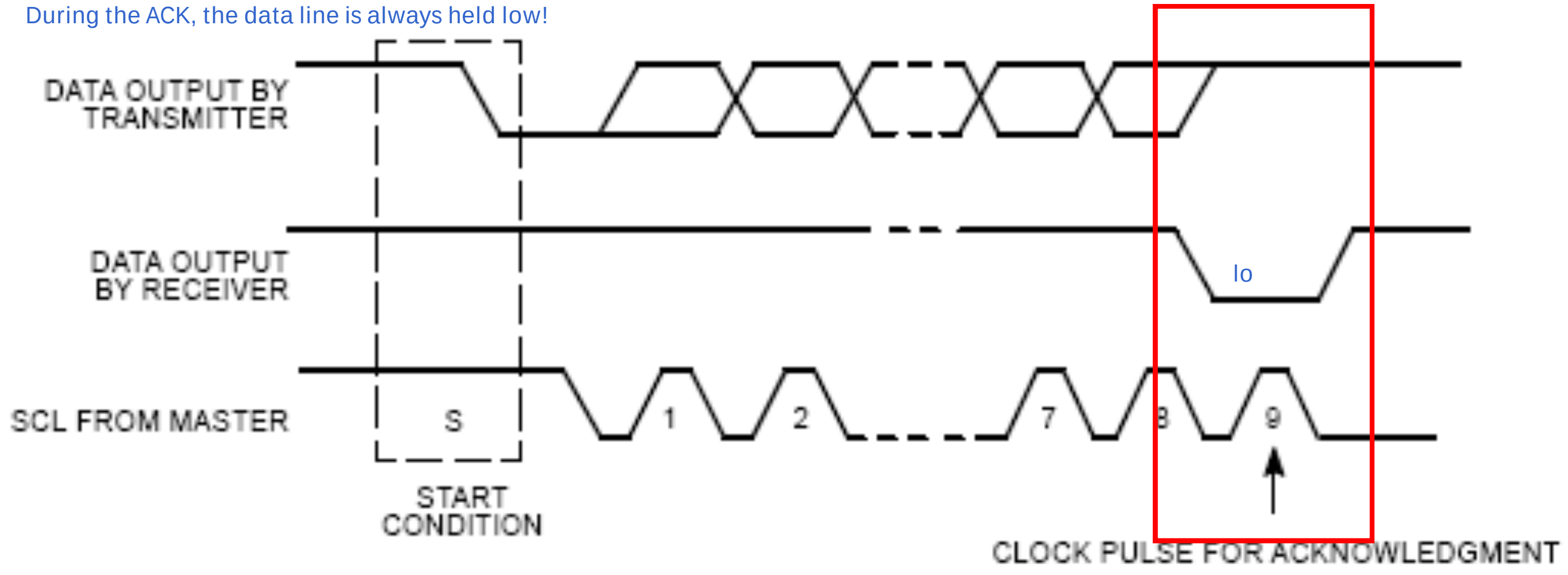
SDA
MSB
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
SCL
S
1
2
7
8
9
ACK
1
2
3 – 8
9
ACK
P
START CONDITION
BYTE COMPLETE, INTERRUPT WITHIN RECEIVER
CLOCK LINE HELD LOW WHILE INTERRUPTS ARE SERVICED
STOP CONDITION

Serial 7N1, 7 data bit, no parity, 1 stop bit
Serial 8E2, 8 useful bit, even parity, 2 stop bits.

Acknowledgment

- Data transfer with ack is obligatory There has to be an ACK related clock pulse
- Ack-related clock pulse is generated by the master
- Transmitter releases the SDA line (HIGH) during the ack clock pulse
- Receiver pulls down the SDA line during the ack clock pulse so that it remains LOW during the HIGH period of this clock pulse

ACK is required. It is repulsory.
The clock pulse for the ACK is essential.
But parity or check bits can be optional.
During the ACK, the data line is always held low!



The ninth clock pulse appears on the SCL line.
ACK is a terminal bit for a byte of data transferred.

Addressing

Someway of knowing who the receivers are. Because this is a party bus, you need to tell which driver you are talking.

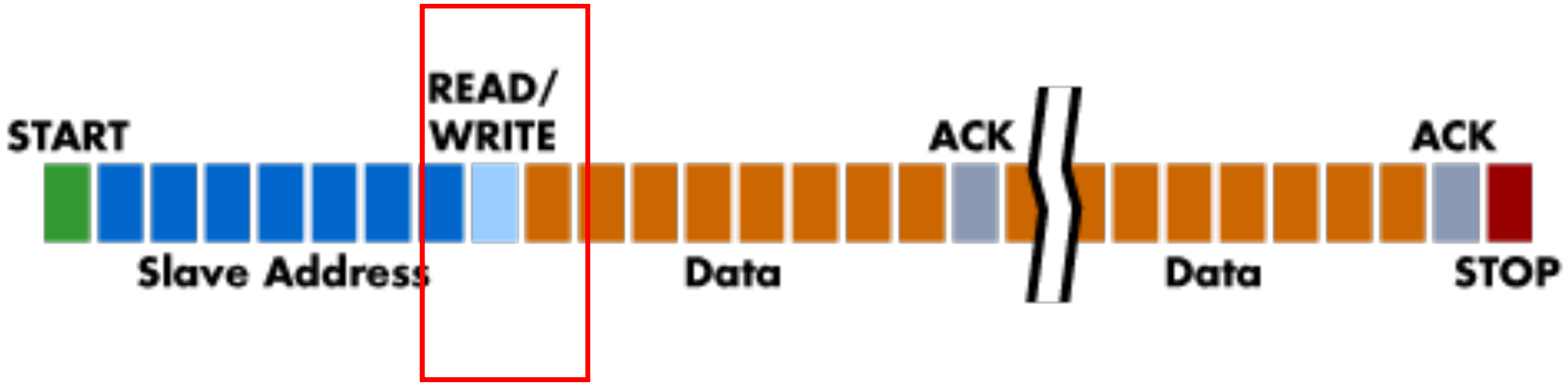
- First byte of transfer contains the slave address and the data direction
  - Address is 7 bits long, followed by the direction bit
  - Like all data bytes, address is transferred with the most significant bit first
- 7-bit address space allows for 128 unique I2C device addresses
  - 16 addresses are reserved for special purposes

    2^7 = 128, 16 special address, 112 address available.
  - Leaves only 112 addresses with this 7-bit address scheme
- New 10-bit address scheme has been introduced

  To get over limitation, in case if you need more than 112 devices.
- "General call" broadcast – to address every device on the bus
- What is the maximum number of devices in I2C limited by?

The bus capacitance -- 112.

Maximum number is technically 112, however, if you have many many devices, the physical bus capacitane will be affecting the many devices, physics limitations.



Clock Stretching

One of the big advantages of I2C.

- Form of flow control
- An addressed slave device may hold the clock line low after receiving (or sending) a bit, indicating that it is not yet ready to process more data
- Master that is communicating with the slave will attempt to raise the clock to transfer the next bit, but
  - If the slave is clock stretching, the clock line will still be low
- Mechanism allows receivers that cannot keep up with a transmitter to control the flow of incoming data

It is completely possible that a receiver device can hold the line low (SCL low), the master wants to hold the line low.
Because it is AND'd, the receiver dictate the clock speed. It can holdo I the clock for so long and consume the data.
The single receiver can also hog the data line -- only it is using itand others cannot access.

In I2C, receiver can dictate the bus speed.

SMBus & PMBus

Intel uses SMBus and PMBus to monitor things on a PC bus.

- SMBus is a derivative of I2C
- Standard developed by Intel and now maintained by the SBS Forum
- Used to monitor critical parameters on PC motherboards and in embedded systems
  - Examples: Supply voltage monitor, temperature monitor, and fan monitor/control ICs with SMBus interfaces
- SMBus differs from I2C in that it incorporates
  - Packet Error Checking (PEC)
  - Timeout for transfers and clock stretching

    So that slave cannot stretch and chock the line forever.
    This mitigate the disadvantages of I2C in clock stretching.
  - ALERT line
  - SUSPEND line
  - Data-transfer formats
  - Maximum bitrate of 100 kHz/s
- PMBus standard is a new protocol on top of the SMBus
  - Defines commands and data structures for power control and management

Summary of Advantages

- Two-wire protocol
  - Minimizes interconnections and pins
  - Cost and power savings
- Multi-master capability
  - Allows for rapid testing
- Easy path for upgrades to embedded systems
  - Just clip on more I2C-compatible devices
  - I2C-compatible peripherals can also be added/removed while the system is running, which makes hot swapping possible

- The beauty is that a microcontroller can control a network of device chips with just two general-purpose I/O pins and software

Simple two line.

I2C came from serial, and evolved from Serial.
SMBus and PMBus built from I2C.

I2C is hot swapping possible.

I2C can be hot plugged. You don't have to power down the entire system to add another I2C device.

I2C is ubiquitous.

Limitations

- Seven bits is not enough to prevent address collisions between the many thousands of available devices Physically limited by the 'bus capacitance', pass a certain point you cannot get voltage levels... etc.
  - Vendors rarely dedicate pins to configure the full address used on a given board
  - Vendors often provide pins to configure a few low order bits of the address and arbitrarily set the higher order bits to some value based on the model
    - Limits the number of devices of that model that can be present on the same bus
  - Ten-bit I2C addresses have not really caught on yet
- Supports a limited range of speeds
  - Bus capacitance also places a limit on the transfer speed
- The dark side of clock stretching
  - Can starve bandwidth needed by faster devices and increase latencies when talking to other devices
- So, typical practice is to have a number of I2C bus *segments*, each with a dozen devices and for a dedicated purpose

  Segmented control either by speed or capacity.
  Dedicated segmented control on low speed I2C / low speed I2C.
  - Example: One segment for high-speed devices, another for power management

I2C can be multi-master. Thus it can be used between MCUs.
But SPI is only single master, that means it's used more between master-slave configuration between MCU and sensors.

Serial Peripheral Interconnect (SPI)

- Another kind of serial protocol in embedded systems (proposed by Motorola)
- Shorthand for "Serial Peripheral Interface"
- Generally faster than I^2C, capable of several Mbps

Applications:

- Like I^2C, used in EEPROM, Flash, and real time clocks
- Better suited for "data streams", i.e., analog-to-digital converters
- Full duplex capability: Communication between a codec and digital signal processor

Serial Peripheral Interconnect (SPI)

- Another kind of serial protocol in embedded systems (proposed by Motorola)

Philips -> I2C
Intel -> SM/PM bus
Motorola -> SPI

- Four-wire protocol
  - SCLK — Serial Clock
  - MOSI/SIMO — Master Output, Slave Input
  - MISO/SOMI — Master Input, Slave Output
  - SS — Slave Select
- Single master device and with one or more slave devices

While I2C can have multuple master devices.

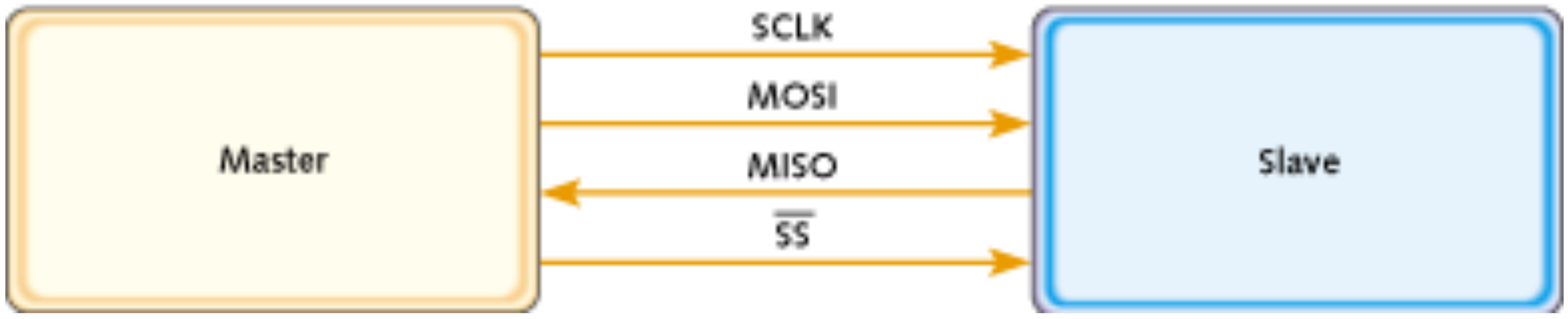
- Higher throughput than I2C and can do "stream transfers"
- No arbitration required
- But
  - Requires more pins
  - Has no hardware flow control
  - No slave acknowledgment (master could be talking to thin air and not even know it)

Serial Peripheral Interconnect (SPI)

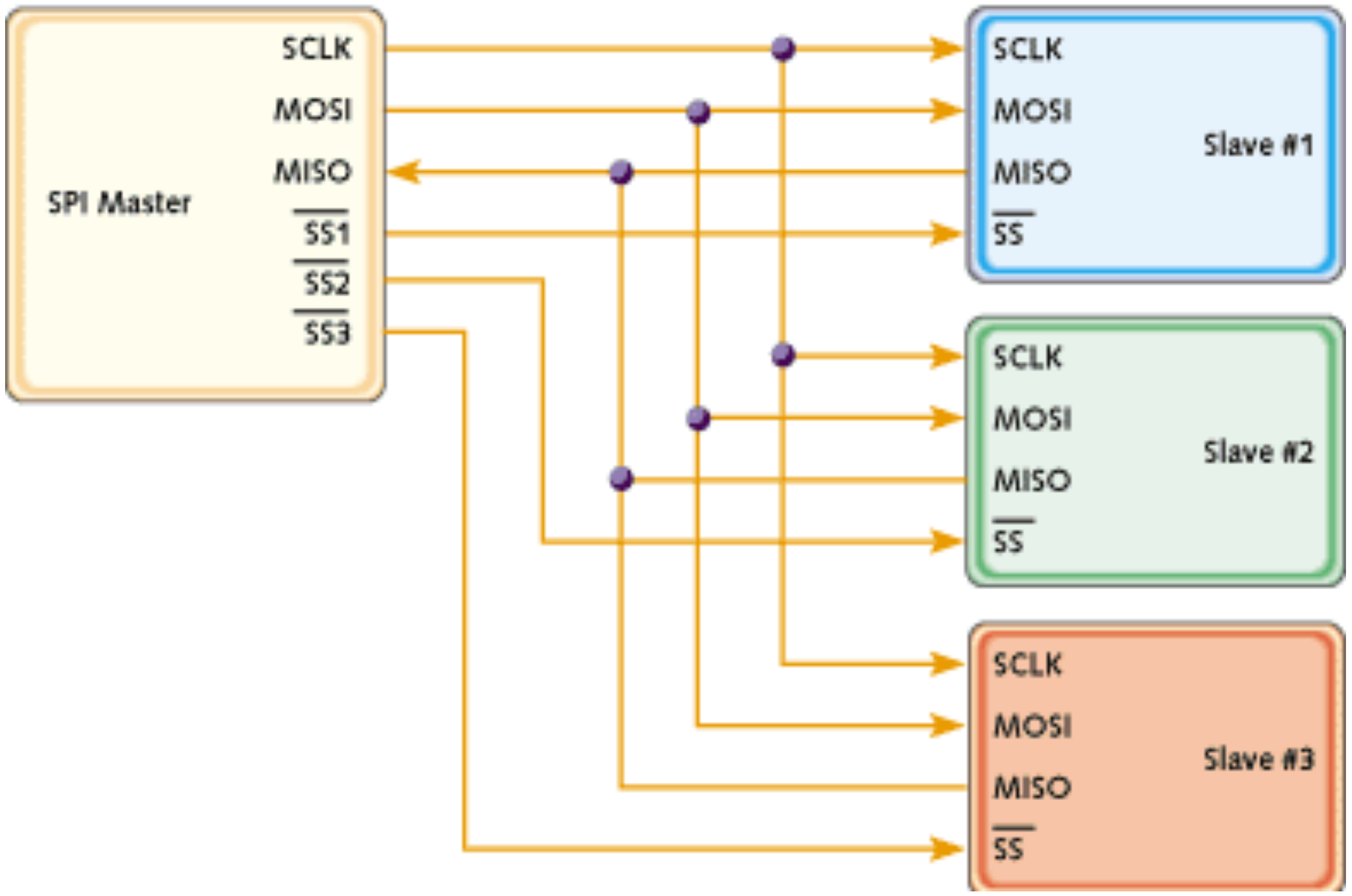
- Synchronous serial data link operating at full duplex
- For point-to-point, SPI is simple and efficient
  - Less overhead than I^2C due to lack of addressing, plus SPI is full duplex
- For multiple slaves, each slave needs separate slave select signal
  - More effort and more hardware than I^2C

SPI there is no flow control, no ACK to confirm receive of data.

Full Duplex means you can be transmitter and receiver at the same time.



Serial Peripheral Interconnect (SPI)



Serial Peripheral Interconnect (SPI)

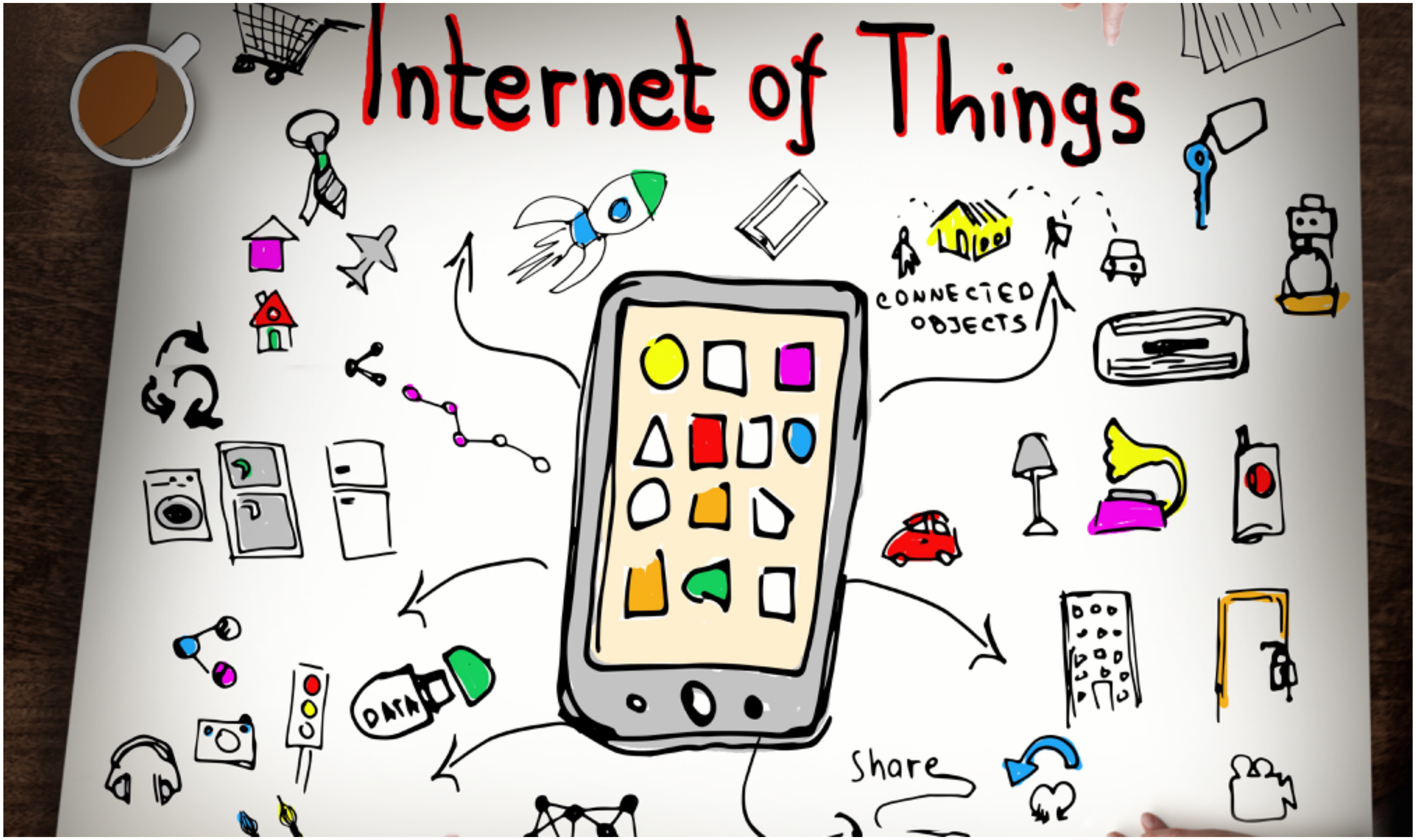
- SPI interface defines only the communication lines and the clock edge
- There is no specified flow control

There is no acknowledgement.

- No acknowledgement mechanism to confirm receipt of data
- For multiple slaves, must employ “bit-banging”

I2C and SPI

- I^{2}C and SPI provide good support for communication with slow peripheral devices that are accessed intermittently
- Serial communication protocols
- Meant for short distances "inside the box" "TV set example, all the wires and boards on in the tv set box."
- Low complexity
- Low cost
- Low speed (a few Mbps at the fastest)
- Mainly EEPROMs and real-time clocks
- I^{2}C easily accommodates multiple devices on a single bus
- SPI is faster, but gets complicated when there is more than one slave involved.



IoT Protocols: I2C, SPI

18-738 Sports Technology

Priya Narasimhan
ECE Department
Carnegie Mellon University
@yinzcampriya